Addressing Security Challenges in Pervasive Computing Applications

Final Technical Report

Contract No. FA9550-07-1-0042

Colorado State University

Principal Investigators

Principal Investigators
Dr. Indrakshi Ray
Dr. Indrajit Ray

Executive Summary

With the growth of mobile and sensor devices, embedded systems, and communication technologies, we are moving towards an era of pervasive computing. This project investigates some of the security challenges of pervasive computing and suggests possible solutions.

Pervasive computing uses numerous, casually accessible, often invisible computing and sensor devices, that are frequently mobile or embedded in the environment and that are interconnected to each other with wired or wireless technology. Being embedded in the environment and strongly interconnected, allows pervasive computing to provide novel services and functionalities that use the knowledge of the surrounding physical spaces. However, it also brings novel security challenges to this new paradigm that can have very serious consequences. Thus, we need to understand the major security and privacy challenges and address these before pervasive computing technology can be widely deployed.

Pervasive computing applications present some unique constraints that preclude the use of traditional security policies and mechanisms from protecting such applications. First, pervasive computing applications typically involve many disparate entities, belonging to different organizations and interacting in complex and subtle ways. Second, the applications are very dynamic in nature with the entities and their interactions potentially changing at any given time. Third, pervasive computing applications use contextual information to provide better services; such information are often used by security mechanisms as well and, hence, must be adequately protected. Fourth, pervasive systems often involve devices with various computation and communication capabilities. Many of these are severely resource constrained, preventing execution of standard security mechanisms on them. The objective of this work is to address some of the security challenges that arise because of these constraints. This work focuses on four major aspects of security in pervasive computing that we summarize below.

Policy and Trust Models for Pervasive Computing Applications

Traditional access control models, such as, Discretionary Access Control (DAC), Mandatory Access Control, and Role-Based Access Control (RBAC), do not use contextual information, namely, space and time, for authorization. Towards this end, we propose a number of increasingly refined spatio-temporal RBAC models where the access decisions depend on the role of the user, her location, the object's location, and the time of access. The models that we develop have

a very sophisticated set features that allow them to express many different spatio-temporal access control constraints. However, they can also interact in many subtle yet complex ways. We show how these features can be formally analyzed to study their interactions. We propose using an automated tool called Alloy, that has embedded SAT-solvers. We have used this model to specify the access control policies of a real-world system – the Dengue Decision Support System that has been developed at Colorado State University. Further, to accommodate the dynamic nature of pervasive computing applications, we propose a graph-theoretic framework to represent the spatio-temporal access control model. This framework allows us to reason about security of the access control configuration changes during application execution.

Pervasive computing applications involve interaction among various entities not all of which are equally trusted. The nature of interactions between entities depend on the trust relationship between them. Towards this end, we developed a new model of trust to characterize and quantify these trust relationships. Our model is based on subjective logic and allows one to reason about the uncertainty that arises in these interactions within pervasive computing applications. We demonstrate the use of this trust model for providing trust-based access control, finding a reliable path for propagating sensor data to processing nodes, and giving sensitive personal data to recipients over an untrusted network.

Designing Secure Pervasive Computing Applications

Pervasive computing applications are inherently complex. They must satisfy functional and non-functional requirements, such as, security. Security cannot be added as an afterthought but must be addressed from the very early stages of design. We demonstrate how aspect-oriented methodology can be used for designing secure applications. In this approach, the application is decomposed into modules on the basis of functionality and the security mechanisms are represented as aspects. We demonstrate how to methodically integrate the security aspects with the functional modules resulting in a design where security requirements have been adequately addressed. Often times, the same security requirement can be satisfied by different security solutions. The solutions may differ with respect to the amount of protection offered, time-to-market, budget and resource constraints. Trade-off analysis must be done to determine which solution best meets the project goal. We propose a new approach to do trade-off analysis that uses Bayesian Belief Networks.

The Unified Modeling Language (UML) is the de facto software specification language used in the industry. We thus use UML for specifying the application and its security constraints. The models must be formally analyzed to provide assurance of correct behavior. Moreover, the analysis must be automated to the extent possible so as to reduce human errors. UML does not have much tool support for automated analysis. Towards this end, we propose a new tool and methodology by which UML specifications can be automatically converted into Alloy. We show how the resulting specification can be evaluated by the Alloy Analyzer. We also show how existing tools for analyzing UML designs, such as, OCLE and USE, can be enhanced to support our analysis.

Security Management in Pervasive Computing Environments

Pervasive computing applications typically involve cooperation among a number of entities spanning multiple organizations. Thus, a security breach can have very far reaching consequences. Moreover, the resource constraints in pervasive environments preclude the use of strong security mechanisms in such applications. Towards this end, we propose a model that can evaluate the chances of an attack occurring. In the event that an attack caused by a malicious worm occurs, it is important to identify the source of attack. The existing practices of fending off such malicious worms are all based on filtering techniques that use signatures derived from the worm code. This may not be fast enough in a pervasive environment. We develop an automatic distributed monitoring system to trace rapidly spreading worms back to their origins.

Pervasive computing applications typically involve information flow over a complex network of devices. The choice of security mechanisms in pervasive environments is influenced by a number of factors, the most important among which are the heterogeneity of the computing devices, resource constraints of these devices, the cost of deploying security mechanisms on these devices, and the attack coverage provided by them. An optimal set of security measures is often difficult to define because of the conflict between the level of security achievable by a mechanism and these other factors. We investigate the problem of selecting a subset of security hardening measures so as to be within a fixed budget and yet minimize the residual damage to the system caused by not plugging all security holes. We refine this model to integrate the attackers perceptions about cost to attack. In a related work, we show how workflow profiles can be used to capture the contexts in which a communication channel can be used in a pervasive environment. We formulate a set of constrained multi-objective optimization problems that minimize the residual damage and the maintenance cost incurred to keep the workflow secure and running.

Controlled Data Dissemination in Pervasive Computing Environments

Pervasive computing environments involve disseminating data to various entities. We need to limit the disclosure of sensitive data. Specifically, we would like to prevent the linking of sensitive data to any specific individual. Thus, in the *k*-anonymity privacy model, information pertaining to an individual is often suppressed or generalized such that he cannot be distinguished from *k* other individuals. Suppressing or generalizing data causes loss of information, which makes the data less useful. We demonstrate how multi-objective optimization can be used to perform a privacy-utility trade-off and give an insight as to whether better privacy is achievable with the same (or nearly same) data utility. Existing privacy models, such as, *k*-anonymity and *l*-diversity, provide a measure of the worst-case privacy but do not capture the privacy-bias that arises because of the anonymization. Towards this end, we propose the use of property vectors to represent privacy and other measurable properties of an anonymization and show how different anonymizations can be compared.

Data availability is also very important in pervasive computing environments. Data access in a pervasive environment can often be modeled by a push-pull based broadcast architecture. In many of these models, the timeliness of servicing the data request becomes critical. Data begin to degrade in utility the later it is provided from a deadline. Thus proper scheduling of the data request is critical to ensure timely availability. We investigate this problem of data broadcast scheduling in an environment where the time criticality is specified by a soft deadline that is directly related to the data utility. Our experiments reveal that the broadcast schedule generated using heuristics can be improved by hybridizing them with local search techniques. Our experiments further illustrate that an evolution strategy based search technique does even better. The work assumes that each request sent by a client is for one data item only, and that multiple requests sent by a client are handled independently from each other. This assumption is eliminated in our subsequent works where each client requires an ordered set of data items, and the client can start processing as soon as it receives the first data item but cannot complete until it gets all the requested data items. Here again, evolution strategies are used to trade-off between the running time of the real-time scheduler and the quality of schedules generated.

The work done as part of this project has been published in various peer-reviewed journals and conferences. The work also resulted in 3 Ph.D. dissertations. The dissertations and papers resulting from this work are listed below.

List of Publications

Doctoral Dissertations

1. Rinku Dewri, "Multi-criteria Analysis in Modern Information Management", Computer Science Department, Colorado State University, Fort Collins, USA, Summer 2010.

2. Nayot Poolsappasit, "Towards an Efficient Vulnerability Analysis Methodology for Better Security Risk Management", Computer Science Department, Colorado State University, Fort Collins, USA, Summer 2010.

3. Manachai Toahchoodee, "Access Control Models for Pervasive Computing Environments", Computer Science Department, Colorado State University, Fort Collins, USA, Summer 2010.

Published Papers

1. Sudip Chakraborty and Indrajit Ray, "p-Trust: A New Model of Trust to Allow Finer Control over Privacy in Peer-to-Peer Framework", *Journal of Computers*, Vol 2(2), April 2007.

2. Indrakshi Ray and Manachai Toahchoodee, "A Spatio-Temporal Role-Based Access Control Model", *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Redondo Beach, CA, July 2007.

3. Sudip Chakraborty, Nayot Poolsappasit and Indrajit Ray, "Reliable Delivery of Event Data from Sensors to Actuators in Pervasive Computing Environments", *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Redondo Beach, CA, July 2007.

4. Lijun Yu, Robert B. France, Indrakshi Ray and Kevin Lano, "A Light-Weight Static Approach to Analyzing UML Behavioral Properties", *Proceedings of the 12th International Conference on Engineering of Complex Computer Systems*, Auckland, New Zealand, July 2007.

5. Kyriakos Anastasakis, Behzad Bordbar, Geri Georg and Indrakshi Ray, "UML2Alloy: A Challenging Model Transformation", *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems*, Nashville, TN, September 2007.

6. Rinku Dewri, Nayot Poolsappasit, Indrajit Ray and Darrell Whitley, "Optimal Security Hardening Using Multi-objective Optimization on Attack Tree Models of Networks", *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, VA, October 2007.

7. Eunjee Song, Robert France, Indrakshi Ray and Hanil Kim, "Checking Policy Enforcement in an Access Control Aspect Model", *Proceedings of the International Conference on Convergence Technology and Information Convergence*, Anaheim, CA, November 2007.

8. Andrew Burt, Michael Darschewski, Indrajit Ray, Ramakrishna Thurimella and Hailin Wu, "Origins: An Approach to Trace Fast Spreading Worms to Their Roots", *International Journal of Security and Networks*, Vol. 3(1), January 2008.

9. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "Optimizing On-Demand Data Broadcast Scheduling in Pervasive Computing Environments", *Proceedings of the 11th International Conference on Extending Database Technology*, Nantes, France, March 2008.

10. Rinku Dewri, Indrajit Ray, Indrakshi Ray and Darrell Whitley, "On the Optimal Selection of k in the K-Anonymity Problems", *Proceedings of the 24th IEEE International Conference on Data Engineering*, Cancun, Mexico, April 2008.

11. Manachai Toahchoodee and Indrakshi Ray, "On the Formal Analysis of a Spatio-Temporal Access Control Model", *Proceedings of the 22nd Annual IFIP WG 11.3 Conference on Data and Applications Security*, London, U.K., July 2008.

12. Indrajit Ray, Nayot Poolsappasit and Rinku Dewri, "An Opinion Model for Evaluating Malicious Activities in Pervasive Computing Systems", *Proceedings of the 22nd Annual IFIP WG 11.3 Conference on Data and Applications Security*, London, U.K., July 2008.

13. Rinku Dewri, Darrell Whitley, Indrakshi Ray and Indrajit Ray, "Evolution Strategy Based Optimization of On-Demand Dependent Data Broadcast Scheduling", *Proceedings of the Genetic and Evolutionary Computation Conference*, Atlanta, GA, July 2008.

14. Indrakshi Ray and Manachai Toahchoodee, "A Spatio-Temporal Access Control Model Supporting Delegation for Pervasive Computing Applications", *Proceedings of the 5th International Conference on Trust, Privacy, and Security in Digital Business*, Turin, Italy, September 2008.

15. Indrakshi Ray and Wei Huang, "Increasing Expressiveness of Events using Parameter Contexts", *Proceedings of the 12th East European Conferences on Advances in Databases and Information Systems*, Pori, Finland, September 2008.

16. Rinku Dewri, Darrell Whitley, Indrajit Ray and Indrakshi Ray, "Optimizing Real-Time Ordered Data Broadcasts in Pervasive Environments using Evolution Strategy", *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, Dortmund, Germany, September 2008.

17. Lijun Yu, Robert France and Indrakshi Ray, "Scenario-based Static Analysis of UML Behavioral Properties", *Proceedings of the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems*, Toulouse, France, September 2008.

18. Indrajit Ray and Sudip Chakraborty, "Facilitating Privacy Related Decisions in Different Privacy Contexts on the Internet By Evaluating Trust in Recipients of Private Data", *Proceedings of the 23rd IFIP TC-11 International Information Security Conference*, Milan, Italy, September 2008.

19. Dilum Bandara, Anura Jayasumana and Indrajit Ray, "Key Pre-distribution Based Secure Backbone Design for Wireless Sensor Networks", *Proceedings of the 3rd IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, Montreal, Canada, October 2008.

20. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "Security Provisioning in Pervasive Environments using Multi-objective Optimizations", *Proceedings of the 13th European Symposium on Research in Computer Security*, Malaga, Spain, October 2008.

21. Nayot Poolsappasit and Indrakshi Ray, "Towards Achieving Personalized Privacy for Location-Based Services", *Transactions on Data Privacy*, 2(1), 2009.

22. Indrakshi Ray, Indrajit Ray and Sudip Chakraborty, "An Interoperable Context-Sensitive Model of Trust", *Journal of Intelligent Information Systems*, 32(1), February 2009.

23. Rinku Dewri, Indrakshi Ray, Darrell Whitley and Indrajit Ray, "On the Comparison of Microdata Disclosure Algorithms", *Proceedings of the 12th International Conference on Extending Database Technology*, Saint-Petersburg, Russia, March 2009.

24. Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee and Siv Hilde Houmb, "An Aspect-Oriented Methodology for Designing Secure Applications", *Information and Software Technology – Special Issue on Model Based Development for Secure Information Systems*, 51(5), May 2009.

25. Manachai Toahchoodee, Indrakshi Ray, Kyriakos Anastasakis, Geri Georg and Behzad Bordbar, "Ensuring Spatio-Temporal Access Control for Real-World Applications", *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, Stresa, Italy, June 2009.

26. Lijun Yu, Robert France, Indrakshi Ray and Sudipto Ghosh, "A Rigorous Approach to Uncovering Security Policy Violations in UML Designs", *Proceedings of the 14th International Conference on Engineering of Complex Computer Systems*, Potsdam, Germany, June 2009.

27. Manachai Toahchoodee, Ramadan Abdunabi, Indrakshi Ray and Indrajit Ray, "A Trust-Based Access Control Model for Pervasive Computing Systems", *Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Montreal, Canada, July 2009.

28. Rinku Dewri, Darrell Whitley, Indrajit Ray and Indrakshi Ray, "A Multi-Objective Approach to Data Sharing with Privacy Constraints and Preference Based Objectives", *Proceedings of Genetic and Evolutionary Computation Conference*, Montreal, Canada, July 2009.

29. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "Multi-Objective Evolutionary Optimization in Statistical Disclosure Control", *Advances in Artificial Intelligence for Privacy Protection and Security*, Imperial College Press, August 2009.

30. Manachai Toahchoodee and Indrakshi Ray, "Using Alloy to Analyse a Spatio-Temporal Access Control Model Supporting Delegation", *IET Information Security*, 3(3), September 2009.

31. Manachai Toahchoodee, Xing Xie and Indrakshi Ray, "Towards Trustworthy Delegation in Role-Based Access Control Models", *Proceedings of the 12th Information Security Conference*, Pisa, Italy, September 2009.

32. Rinku Dewri, Indrajit Ray, Indrakshi Ray and Darrell Whitley, "POkA: Identifying Pareto-Optimal k-Anonymous Nodes in a Domain Hierarchy Lattice", *Proceedings of the 18th ACM*

Conference on Information and Knowledge Management, Hong Kong, China, November 2009.

33. Kyriakos Anastasakis, Behzad Bordbar, Geri Georg and Indrakshi Ray, "On Challenges of Model Transformation from UML to Alloy", *Software and Systems Modeling*, 9(1), January 2010.

34. Indrajit Ray, Indrakshi Ray and Sudip Chakraborty, "A Context-Aware Model of Trust for Facilitating Secure Ad Hoc Collaborations", *Trust Modeling and Management in Digital Environments: From Social Concept to System Development*, IGI Global, January 2010.

35. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "Query m-Invariance: Preventing Query Disclosures in Continuous Location-Based Services", Proceedings of the 11th International Conference on Mobile Data Management, Kansas City, Missouri, May 2010.

36. Geri Georg, Kyriakos Anastasakis, Behzad Bordbar, Siv Hilde Houmb, Indrakshi Ray and Manachai Toahchoodee, "Verification and Trade-off Analysis of Security Properties in UML System Models", *IEEE Transactions on Software Engineering*, 36(3), May-June 2010.

37. Rinku Dewri, Indrajit Ray, Indrakshi Ray and Darrell Whitley, "On the Identification of Property Based Generalizations in Microdata Anonymization", *Proceedings of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Rome, Italy, June 2010.

38. Siv Hilde Houmb, Sudip Chakraborty, Indrakshi Ray and Indrajit Ray, "Using Trust-Based Information Aggregation for Predicting Security Level of Systems", *Proceedings of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Rome, Italy, June 2010.

39. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "Real Time Stochastic Scheduling in Broadcast Systems with Decentralized Data Storage", *Real-Time Systems*, 45(3), August 2010.

40. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "On the Formation of Historically *k*-Anonymous Anonymity Sets in a Continuous LBS", *Proceedings of the 6th International Conference on Security and Privacy in Communication Networks*, Singapore, September 2010.

41. Rinku Dewri, Indrakshi Ray, Indrajit Ray and Darrell Whitley, "*k*-Anonymization in the Presence of Publisher Preferences", To appear in *IEEE Transactions on Knowledge and Data Engineering*.

42. Siv Hilde Houmb, Geri Georg, Dorina C. Petriu, Behzad Bordbar, Indrakshi Ray, Kyriakos Anastasakis and Robert France, "Balancing Security and Performance Properties During System Architectural Design", To appear in *Software Engineering for Secure Systems: Industrial and Research Perspectives*.

43. Manachai Toahchoodee and Indrakshi Ray, "On the Formalization and Analysis of a Spatio-Temporal Role-Based Access Control Model", To appear in *Journal of Computer Security*.

Publications under Review

1. Rinku Dewri, Indrajit Ray, Indrakshi Ray and Darrell Whitley, "Exploring Privacy versus Data Utility Tradeoffs in Anonymization Techniques using Multi-objective Optimization", *Journal of Computer Security* (submitted first revision).

2. Siv Hilde Houmb, Indrakshi Ray, Indrajit Ray and Sudip Chakraborty, "Trust-Based Security Level Evaluation using Bayesian Belief Networks", *Transactions on Computational Sciences Journal* (submitted first revision).

3. Nayot Poolsappasit, Rinku Dewri and Indrajit Ray, "Dynamic Security Risk Assessment and Mitigation Using Bayesian Attack Graphs", *IEEE Transactions on Dependable and Secure Computing* (initial submission).

4. Rinku Dewri, Indrajit Ray, Nayot Poolsappasit and Darrell Whitley, "Optimal Security Hardening on Attack Tree Models of Networks: A Cost-Benefit Analysis", *International Journal of Information Security* (initial submission).

5. Rinku Dewri, Indrajit Ray, Indrakshi Ray and Darrell Whitley, "Utility Driven Optimization of Real Time Data Broadcast Schedules", *Applied Soft Computing* (initial submission).

6. Manachai Toahchoodee, Indrakshi Ray and Ross McConnell, "Using Graph Theory to Represent a Spatio-Temporal Role-Based Access Control Model", *International Journal of Next-Generation Computing* (initial submission).

List of Personnel

Personnel Supported by Grant

The work of the following personnel were partially supported by this grant. 1. Rinku Dewri (Student) 2. Nayot Poolsappasit (Student) 3. Indrakshi Ray (Faculty) 4. Indrajit Ray (Faculty) 5. Manachai Toahchoodee (Student)

Contents

List of Figures

Chapter 1

Introduction

In order to win our nation's wars in the new millennium the U.S. Air Force plans to transform itself into a net-centric, knowledge-based force. Pervasive computing is an emerging paradigm that has the potential to act as an enabler for this goal. Pervasive computing uses numerous, casually accessible, often invisible computing and sensor devices, that are frequently mobile or embedded in the environment and that are interconnected to each other with wireless or wired technology. Being embedded in the environment and strongly interconnected, allow pervasive computing devices to exploit knowledge about the operating environment in a net-centric manner. Thus they provide a rich new set of services and functionalities that are not possible through conventional means.

Although pervasive computing technology looks promising, one critical challenge needs to be addressed before it can be widely deployed – security. The very knowledge that enables a pervasive computing application to provide better services and functionalities may easily be misused, causing security breaches. The problem is serious because pervasive computing applications involve interactions between a large number of entities that can span different organizational boundaries. Unlike traditional applications, these applications do not usually have well-defined security perimeter and are dynamic in nature. Moreover, these applications use knowledge of surrounding physical spaces. This requires security policies to use contextual information that, in turn, must be adequately protected from security breaches. Uncontrolled disclosure of information or unconstrained interactions among entities can lead to very serious consequences. Traditional security policies and mechanisms rarely address these issues and are thus inadequate for securing pervasive computing applications. Our work focuses on understanding the security challenges involved in pervasive computing applications and proposing solutions to some of the problems.

In subsequent paragraphs we summarize the various aspects related to security of pervasive computing environments that we investigated in this project. Details about these works can be found in our publications. We highlight some of the more important contributions in the remaining chapters of this report.

Our first task focussed on access control models for pervasive computing applications. Although a lot of research appears in extending traditional access control models for novel applications, we found the models not expressive enough to meet the requirements of pervasive computing applications. Our first contribution involves extending existing access control models to incorporate the notion of location and time. Our models allow the application to specify various types of spatio-temporal constraints that may arise in pervasive computing applications. The various features of the models may interact resulting in conflicts and inconsistencies. Towards this end, we show how the models can be formally analyzed. We use the Alloy Analyzer, which has an embedded SAT solver, to understand the subtleties involved with feature interactions. We demonstrate the applicability of this model in a real-world – the Dengue Decision Support system that is being designed at Colorado State University to be deployed in Mexico. An application using our access control model must be analyzed to provide assurance that correct policies have been specified. Towards this end, we show how such analysis can be done using two techniques: one using UML and Alloy and the other using Coloured Petri Nets (CPNs).

Pervasive computing applications are dynamic in nature – the entities, the resources, and the access patterns may change during the course of application. In the face of such dynamism, it is essential to ensure that access control breaches do not occur. Since the required analysis must be done in real-time, it is equally important to minimize the verification time. To address this important problem, we formalize the semantics of our spatio-temporal model using graph theory and provide incremental analysis techniques. We achieve very good complexity results. In addition, one side effect of this work is the development of a new and efficient common predecessor detecting algorithm in a dynamic graph, the results of which can be used in various application domains.

Pervasive computing environments often involve interactions with different types of entities, not all of which are equally trustworthy. The nature of interactions between entities depend on the trust relationship between them. Towards this end, we model and quantify trust relationships within pervasive applications. In the model that we propose, the trust relationship between a truster and trustee is associated with a context and depends on the experience, knowledge, and recommendation that a truster has with respect to the trustee in the given context. Experience quantifies the past interactions that the truster had with the trustee, knowledge assesses the verifiable properties of the trustee, and recommendation measures how much other entities trust the trustee with respect to the given context. The absence of one or more of these values in a given context precludes computing the trust value in that given context. To overcome this problem, we formalize the notion of contexts and capture the relationships between different contexts in the form of a context graph. This allows one to extrapolate trust values from related contexts when all the information needed to compute trust is not available. It also helps resolve the semantic mismatches that occur when various sources use different terminology to represent contexts. We demonstrate the use of this trust model for providing trust-based access control in pervasive computing systems and also for finding a reliable path for propagating sensor data to processing nodes.

Security management is an important task in pervasive computing environments as some devices, specially sensor nodes, have limited computation and communication capabilities. For security management of these applications it is necessary to impose and maintain some secured structure within the sensor network if one is involved in the application. Clustering is a key technique that simplifies network management in such large-scale sensor networks. A secure backbone, built by a cooperating hierarchy of clusters in the form of a cluster tree, can further enhance upper layer functions, such as secure routing, secure session key distribution between applications, secure broadcasting, and secure query delivery. We investigate the design of such a secure backbone for sensor networks based on the cluster tree approach. We integrate the Hierarchical Hop-ahead Clustering algorithm with a secret key pre-distribution scheme to build such a secure backbone. The key pre-distribution scheme based on Random Block Merging in Combinatorial Design has very low computational cost and communication overhead. The protocol ensures that at least one common key exist between any pair of nodes.

The rich connectivity among computing elements in pervasive environments and abundance of low capability devices may cause irreparable damage by an attack. In order to address this problem, we propose a model that evaluates the chances of a successful attack. This allows one to put appropriate security controls where and when needed. In spite of security controls, it is possible for fast spreading worms to wreck havoc. Typically, we protect against such malicious worms using filtering techniques based on signatures derived from the worm code. However, worms can be designed to spread so rapidly that by the time a signature is developed and distributed the damage is done, thus rendering any signature-based mediation futile. We formulate an automatic distributed monitoring system to trace rapidly spreading worms back to their origins. It works by correlating anomalous events across a network and establishing a causal relationship between them. We show that even with less than perfect deployment (about 20%) of this system, it can very rapidly and accurately narrow down the worm origin to a small set of possibilities. Appropriate action can then be taken to respond to such attacks.

Pervasive computing applications typically involve information flow over a complex network of devices. Effective security mechanisms need to be deployed to protect these applications. The choice of security mechanisms in pervasive environments is influenced by a number of factors, the most important among which are the heterogeneity of the computing devices, resource constraints of these devices, the cost of deploying security mechanisms on these devices, and the attack coverage provided by them. An optimal set of security measures is often difficult to define because of the conflict between the level of security achievable by a mechanism and these other factors. As a first step, we investigate the problem of selecting a subset of security hardening measures so as to be within a fixed budget and yet minimize the residual damage to the system caused by not plugging all security holes. We formulate the problem as a multi-objective optimization problem and develop a systematic approach to solve the problem using non-dominated sorting genetic algorithm on an attack tree model of the system. We believe that an attacker's perceived gains through a specific attack strategy can (and should) influence the security administrator's decision to employ a particular defense strategy. Thus we refine the security provisioning problem as a payoff problem to maximize the return on investment under the scenario that an attacker is actively engaged in maximizing its return on attacks. Subsequently, we show how workflow profiles can be used to capture the contexts in which a communication channel can be used in a pervasive environment. We formulate a set of constrained multi-objective optimization problems that minimize the residual damage and the maintenance cost incurred to keep the workflow secure and running.

Pervasive computing applications often involve sharing sensitive data across organizational boundaries. For instance, one may want to prevent disclosing the identity of an individual. One well-known model preventing identity disclosure is the k-anonymity model. The idea is to make a tuple indistinguishable from $k-1$ other tuples by generalizing and/or suppressing attributes. Unfortunately, such transformations result in a considerable loss of information. The information loss is proportional to the value of k. Studies have focussed on minimizing the information loss for some given value of k. However, owing to the presence of outliers, a specified k value may not be obtainable all the time. Further, an exhaustive analysis is required to determine a k value that fits the loss constraint acceptable to a data requester. We investigate the problem of finding an optimal value of k for a given data set. Specifically, we develop a methodology to analyze the trade-off of the generalization losses involved with variations in k. Such types of analysis can reveal, for example, that it is possible to provide a higher level of privacy for a higher fraction of the data set without compromising much on its information content. It can also identify ways of examining if the level of privacy required by a human subject is achievable within the acceptable limits of perturbing data quality. We use multi-objective evolutionary optimization for exploring the trade-offs involved with minimizing information loss and maximizing privacy.

Privacy models, such as k-anonymity, offer an aggregate or scalar notion of the privacy property that holds collectively on the entire anonymized data set. However, they fail to give an accurate measure of privacy with respect to the individual tuples. For example, two anonymizations achieving the same value of k in the k-anonymity model will be considered equally good with respect to privacy protection. However, it is possible that in one anonymization a majority of individuals have a higher probability of privacy breach than the other. We, therefore, reject the notion that all anonymizations satisfying a particular privacy property, such as k-anonymity, are equally good. The scalar or aggregate value used in the privacy models is often biased towards a fraction of the data set, resulting in higher privacy for some individuals and minimal for others. To better compare anonymization algorithms, there is a need to formalize and measure this bias. Towards this end, we propose the use of property vector to represent privacy and other measurable properties of an anonymization. We show how anonymizations can be compared using quality index functions that quantify the effectiveness of property vectors. We also propose some preference based techniques when comparisons must be made across multiple properties induced by anonymizations.

Data availability is also very important in pervasive computing applications. Data access in a pervasive environment can be modeled by a push-pull based broadcast architecture, specifically characterized by the time critical nature of the data requests. We investigate the problem of data broadcast scheduling in an environment where the time criticality is specified by a soft deadline that is directly related to the data utility. Our experiments reveal that the broadcast schedule generated using heuristics can be improved by hybridizing them with local search techniques. Our experiments further illustrate that an evolution strategy based search technique does even better.

Pervasive computing applications are very complex. Security issues cannot be added as an afterthought in such applications. We demonstrate how to design such applications using an aspect-oriented methodology. In our approach, the application is decomposed into modules on the basis of functionality – we refer to this as the primary model. We model each security concern that is of interest as an aspect. The aspect is then methodically composed with our primary model. The result of the composition is a model that represents the application in which the security concern has been addressed. We show how to verify resulting model to ensure that the important properties of aspects are preserved in it. We also demonstrate how to do trade-offs among different security aspects all of which satisfy the same security property by using Bayesian Belief Networks.

Since the Unified Modeling Language (UML) is the de facto specification language in the software industry, we use it to for modeling the aspects and primary model. However, UML does not have much tool support for automated analysis. Towards this end, we show how existing tools for UML analysis, such as OCLE and USE, can be extended to support behavioral analysis. We also demonstrate an alternative approach that involves converting the UML specification automatically to Alloy using UML2Alloy and verify the resulting specification using the Alloy Analyzer.

The rest of the report highlights some of our more important contributions. It is organized as follows. Chapter 2 presents our spatio-temporal role-based access control model that can be used for pervasive computing applications. Chapter 3 demonstrates the use of this model for real-world applications and shows how to provide assurance that no access control breach occurs. Chapter 4 refines the model and expresses the semantics using graph-theory. Chapter 5 proposes a trust model, based on subjective logic, that can be used for pervasive computing applications. Chapter 6 describes how risk estimation and security provisioning can be done in the face of resource constraints. Chapter 7 shows how location information, captured by pervasive computing applications, can be disseminated in a careful and controlled manner. Chapter 8 provides a methodology for designing secure pervasive computing applications. Chapter 9 concludes this report and gives some future directions.

Chapter 2

Spatio-Temporal Role-Based Access Control Model

Pervasive computing applications use the knowledge of the surrounding context to provide better applications and services. Context information can be also used to provide better security for such applications. For example, access to a system need to be enabled only when a user enters a room and it to be disabled when he leaves the room. Traditional access control models, such as, DAC, BLP, or RBAC, do not take into account such environmental factors while making access decisions. Towards this end, we propose a spatio-temporal access control model for use in pervasive computing applications.

We choose to base our model on RBAC primarily because the latter is policy-neutral, simplifies access management, and widely used by commercial applications. We illustrate how each component of RBAC can be related with time and location, and explain how they impact each entity and relationship in RBAC. We also demonstrate how spatio-temporal information can be used for making access decisions. The various features supported by our model are specified using logical constraints. These features often interact in subtle ways resulting in inconsistencies and conflicts. Consequently, it is important to analyze and understand these interactions before such models can be widely deployed.

Manual analysis is often not rigorous, frequently tedious and error-prone. Analyzers based on theorem proving are hard to use, require expertise, and need manual intervention. Model checkers are automated but are limited by the size of the system they can verify. Considering these, we advocate the use of Alloy [24] for checking access control models. Alloy is a modeling language capable of expressing complex structural constraints and behavior. It supports automated analysis. Moreover, it has been successfully used in the modeling and analysis of real-world systems [15, 48]. We demonstrate how Alloy can be used for analyzing the interaction of the different features of our access control model.

The rest of the chapter is organized as follows. Section 2.1 describes the highlights of our model. Section 2.2 illustrates our analysis techniques using Alloy. Section 2.3 concludes this chapter with directions for future work.

2.1 Our Model

We briefly describe how the different entities of core RBAC, namely, *Users*, *Roles*, *Sessions*, *Permissions*, *Objects* and *Operations*, can be associated with location and time. This forms the basis of our new authorization model.

Users and Objects

Users in our model can be human users or other entities such as sensor devices. For the rest of this discussion we refer to a user as a human user although all the concepts presented here applies equally to other entities. We assume that each valid user, interested in doing some location-sensitive operation, carries a locating device which is able to track her or its location. The location of a user changes with time. The relation $UserLocation(u,t)$ gives the location of the user at any given time instant t. Since a user can be associated with only one location at any given point of time, we have the following constraint:

$$UserLocation(u,t) = l_i \wedge UserLocation(u,t) = l_j \Leftrightarrow (l_i \subseteq l_j) \vee (l_j \subseteq l_i)$$

We define a similar function $UserLocations(u,d)$ that gives the location of the user during the time interval d. We define a function $ObjLocations(o,d)$ in the same manner which gives the location of an object at any given time.

Roles

We have three types of relations with roles. These are user-role assignment, user-role activation, and permission-role assignment. We begin by focusing on user-role assignment. In our model, a user must satisfy spatial and temporal constraints before roles can be assigned. We capture this with the concept of *role allocation*. A role is said to be *allocated* when it satisfies the temporal and spatial constraints needed for role assignment. A role can be assigned once it has been allocated. $RoleAllocLoc(r)$ gives the set of locations where the role can be allocated. $RoleAllocDur(r)$ gives the time interval where the role can be allocated. Some role s can be allocated anywhere, in such cases $RoleAllocLoc(s) = universe$. Similarly, if role p can be assigned at any time, we specify $RoleAllocDur(p) = always$.

Some roles can be activated only if the user is in some specific locations at a given time. We borrow the concept of *role-enabling* [4, 29] to describe this. A role is said to be *enabled* if it satisfies the temporal and location constraints needed to activate it. A role can be activated only if it has been enabled. $RoleEnableLoc(r)$ gives the location where role r can be activated and $RoleEnableDur(r)$ gives the time interval when the role can be activated.

The predicate $UserRoleAssign(u,r,d,l)$ states that the user u is assigned to role r during the time interval d and location l. For this predicate to hold, the location of the user when the role was assigned must be in one of the locations where the role allocation can take place. Moreover, the time of role assignment must be in the interval when role allocation can take place.

$$UserRoleAssign(u,r,d,l) \Rightarrow (UserLocation(u,d) = l) \wedge \\ (l \subseteq RoleAllocLoc(r)) \wedge (d \subseteq RoleAllocDur(r))$$

The predicate $UserRoleActivate(u,r,d,l)$ is true if the user u activated role r for the interval d at location l. This predicate implies that the location of the user during the role activation must be a subset of the allowable locations for the activated role, all time instances when the role remains activated must belong to the duration when the role can be activated, and the role can be activated only if it is assigned.

$$UserRoleActivate(u,r,d,l) \Rightarrow \\ (l \subseteq RoleEnableLoc(r)) \wedge (d \subseteq RoleEnableDur(r)) \wedge UserRoleAssign(u,r,d,l)$$

The additional constraints imposed upon the model necessitates changing the preconditions of the functions $AssignRole$ and $ActivateRole$.

Permissions

The goal of our model is to provide better security than their traditional counterparts. This happens because the time and location of a user and an object are taken into account before making the access decisions. Our model also allows us to model real-world requirements where access decision is contingent upon the time and location associated with the user and the object.

Permissions are associated with roles, objects, and operations. We associate three additional entities with permission to deal with spatial and temporal constraints: user location, object location, and time. We define three functions to retrieve the values of these entities. $PermRoleLoc(p,r)$ specifies the allowable locations that a user playing the role r must be in for him to get permission p. $PermObjLoc(p,o)$ specifies the allowable locations that the object o must be in so that the user has permission to operate on the object o. $PermDur(p)$ specifies the allowable time when the permission can be invoked.

We define another predicate which we term *PermRoleAcquire(p,r,d,l)*. This predicate is true if role r has permission p for duration d at location l. Note that, for this predicate to be true, the time interval d must be contained in the duration where the permission can be invoked and the role can be enabled. Similarly, the location l must be contained in the places where the permission can be invoked and role can be enabled.

$$PermRoleAcquire(p,r,d,l) \Rightarrow (l \subseteq (PermRoleLoc(p,r) \cap RoleEnableLoc(r))) \\ \wedge (d \subseteq (PermDur(p) \cap RoleEnableDur(p)))$$

The predicate $PermUserAcquire(u,o,p,d,l)$ means that user u can acquire the permission p on object o for duration d at location l. This is possible only when the permission p is assigned some role r which can be activated during d and at location l, the user location and object location match those specified in the permission, the duration d matches that specified in the permission.

$$PermRoleAcquire(p,r,d,l) \wedge UserRoleActivate(u,r,d,l)$$
$$\wedge(ObjectLocation(o,d) \subseteq PermObjectLoc(p,o)) \Rightarrow PermUserAcquire(u,o,p,d,l)$$

Impact of Time and Location on Role-Hierarchy

Organization structure is reflected in RBAC in the form of a role hierarchy [45] which is a transitive, anti-symmetric relation among roles. Senior roles can inherit the permissions of junior roles, or a senior role can activate a junior role, or do both depending on the nature of the hierarchy. Joshi et al. [29] identify two basic types of hierarchy. The first is the permission inheritance hierarchy where a senior role x inherits the permission of a junior role y. The second is the role activation hierarchy where a user assigned to a senior role can activate a junior role. Each of these hierarchies may be constrained by location and temporal constraints. Consequently, we have a number of different hierarchical relationships in our model one of which is described below.

[Unrestricted Permission Inheritance Hierarchy] A senior role inherits the junior roles permissions but not the spatial and temporal constraints associated with it. If x and y are roles such that $x \geq y$, that is, senior role x has an unrestricted permission-inheritance relation over junior role y, then x inherits y's permissions but not the locations and time associated with it.

$$(x \geq y) \wedge PermRoleAcquire(p,y,d,l) \Rightarrow PermRoleAcquire(p,x,always,universe)$$

We define the other hierarchies, namely, *unrestricted activation hierarchy*, *location restricted permission inheritance hierarchy*, *location restricted activation hierarchy*, *time restricted permission inheritance hierarchy*, *time restricted activation hierarchy*, *time location restricted permission inheritance hierarchy*, and *time location restricted activation hierarchy*, in a similar manner. The hierarchies differ with respect to the spatio-temporal constraints imposed on the corresponding hierarchical relationship.

Impact of Time and Location on Separation Of Duties

Separation of duties (SoD) enables the prevention of the fraud that may be caused by the user [46] when she performs an action that require two or more steps. SoD can be either static or dynamic. Static Separation of Duty (SSoD) comes in two varieties. First one is with respect to user role assignment. The second one is with respect to permission role assignment. In this case, the SSoD constraint is specified as a relation between roles. The idea is that the same user cannot be assigned to the same role. Due to the presence of temporal and spatial constraints, we can have different flavors of separation of duties – some that are constrained by temporal and spatial constraints and others that are not. One example of such a constraint is as follows:

[Weak Form of SSoD - User Role Assignment] Let x and y be two roles such that $x \neq y$. $x, y \in SSOD_w(ROLES)$ if the following condition holds:

$UserRoleAssign(u,x,d,l) \Rightarrow \neg\, UserRoleAssign(u,y,d,l)$

The above definition says that a user u assigned to role x during time d and location l cannot be assigned to role y at the same time and location if x and y are related by $SSOD_w$.

We have other forms of SSoD constraints that we do not elaborate here. These include *strong temporal form of SSoD – user role assignment*, *strong spatial form of SSoD – user role assignment*, *strong form of SSoD – user role assignment*, *weak form of SSoD – permission role assignment*, *strong temporal form of SSoD – user role assignment*, *strong spatial form of SSoD – user role assignment*, and *strong form of SSoD – permission role assignment*. These differ with respect to the influence of spatio-temporal constraints on the relationships. We have various flavors of DSoD constraints as well that are identified in our publications [42, 43, 50, 51, 52].

2.2 Model Analysis

We use Alloy to analyze the interaction of the various features of the access control model. Alloy is supported by an automated constraint solver called Alloy Analyzer that searches instances of the model to check for satisfaction of system properties. The model is automatically translated into a Boolean expression, which is analyzed by SAT solvers embedded within the Alloy Analyzer. A user-specified scope on the model elements bounds the domain, making it possible to create finite Boolean formulae that can be evaluated by the SAT-solver. When a property does not hold, a counter example is produced that demonstrates how the property has been violated.

An Alloy model consists of *signature* declarations, *fields*, *facts* and *predicates*. Each signature consists of a set of *atoms* which are the basic entities in Alloy. Atoms are *indivisible* (they cannot be divided into smaller parts), *immutable* (their properties do not change) and *uninterpreted* (they do not have any inherent properties). Each field belongs to a signature and represents a relation between two or more signatures. A relation denotes a set of tuples of atoms. Facts are statements that define constraints on the elements of the model. Predicates are parameterized constraints that can be invoked from within facts or other predicates.

The basic entities in the access control model, such as, *User*, *Time*, *Location*, *Role*, *Permission* and *Object* are represented as signatures. For instance, the declarations shown below define a set named *User* and a set named *Role* that represent the set of all users and the set of all roles in the system respectively. Inside the *Role* signature body, we have four relations, namely, *RoleAllocLoc*, *RoleAllocDur*, *RoleEnableLoc*, and *RoleEnableDur* which relates *Role* to other signatures.

```
sig User{}
sig Role{
 RoleAllocLoc: Location,
 RoleAllocDur: Time,
 RoleEnableLoc: Location,
 RoleEnableDur: Time}
```

The different relationships between the components in our model are also expressed as signatures. For instance, *RoleEnable* has a field called *member* that maps to a cartesian product of *Role*, *Time* and *Location*. Similarly, *RoleHierarchy* has a field *RHmember* that represents a relationship between *Role* and *Role*. Different types of role hierarchy are modeled as the subsignatures of *RoleHierarchy*.

```
sig RoleEnable { member: Role -> Time -> Location}
sig RoleHierarchy { RHmember: Role -> Role}
sig UPIH, TPIH, LPIH, TLPIH, UAH, TAH, LAH, TLAH extends
     RoleHierarchy{}
```

The various invariants are represented as facts in Alloy. For instance, the fact *URActivate* states that for user *u* to activate role *r* during the time interval *d* and location *l*, this user has to be assigned to role *r* in location *l* during time *d*. Moreover, the location of the user must be a subset of the locations where the role is enabled, and the time must be in the time interval when role *r* can be enabled. This is specified in Alloy as shown below. Other invariants are modeled in a similar manner.

```
fact URActivate{
all u: User, r: Role, d: Time, l: Location, uras: UserRoleAssignment,
urac: UserRoleActivate |
((u->r->d->l) in urac.member) => (((u->r->d->l) in uras.member) &&
(l in r.RoleEnableLoc) && (d in r.RoleEnableDur))
}
```

We use Alloy's *fact* feature to represent the properties of the different hierarchies. The fact *UPIHFact* represents the Unrestricted Permission Inheritance Hierarchy's property. The fact states that senior role `sr` can acquire all permissions assigned to itself together with all permissions assigned to the junior role `jr` .

```
//Unrestricted Permission Inheritance Hierarchy
fact UPIHFact{
     all sr, jr: Role, p: Permission, d: Time, l: Location, upih: UPIH,
         rpa: RolePermissionAssignment, pra: PermRoleAcquire |
         ((sr->jr in upih.member) && (jr->p->d->l in pra.member) &&
        (sr->p !in (rpa.member).Location.Time)) =>
             (sr->p->sr.RoleEnableDur->sr.RoleEnableLoc) in pra.member}
```

The separation of duty constraints are modeled as predicates. Consider the weak form of SSoD User Role Assignment. This constraint says that a user *u* assigned to role *r*1 during time *d* and location *l* cannot be assigned to its conflicting role *r*2 at the same time and location. The other forms are modeled in a separate manner.

```
//Weak Form of SSoD-User Role Assignment
pred W_SSoD_URA(u: User, disj r1, r2: Role,
ura: UserRoleAssignment.member, d: Time, l: Location){
((u->r1->d->l) in ura) => ((u->r2->d->l) not in ura)
}
```

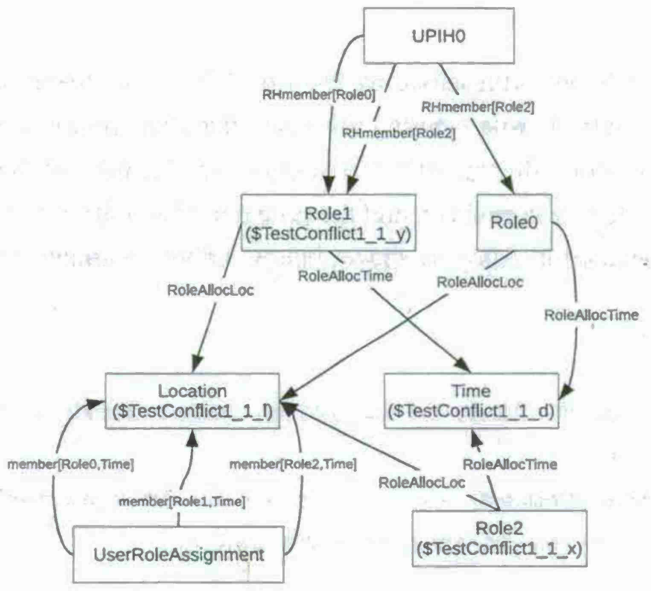


Figure 2.1: Counterexample for assertion *TestConflict*

Once our access control model has been specified in Alloy, we need to verify whether any conflicts occur between the features of the model. We rely on the capabilities of the Alloy analyzer for this purpose. We create an *assertion* that specifies the properties we want to check. Once the assertion has been created, we let Alloy analyzer validate the assertion by using *check* command. If our assertion is wrong in the specified scope, Alloy analyzer will show the counterexample. For instance, to check the interaction of the weak form of SSoD, User Role Assignment and the Unrestricted Permission Inheritance Hierarchy, we make the assertion shown below. The assertion does not hold as illustrated by the counterexample shown in Figure 2.1.

```
// WSSoD_URA violation in the present of UPIH Hierarchy
check TestWSSoD_URA
assert TestConflict{
        no u: User, disj x, y: Role, upih: UPIH,
      d: Time, l: Location, ura: UserRoleAssignment |
                 ((x->y in ^(upih.member)) &&
             (u->x->d->l in ura.member)) =>
                             W_SSoD_URA[u, x, y, u->(x+y)->d->l, d, l]
}
check TestConflict
```

The counterexample shows one possible scenario. In this case, it uses the following instances to show the violation.

1. $Role = \{Role0, Role1, Role2\}$
2. $UPIH0 = \{Role0 \rightarrow Role1, Role2 \rightarrow Role0, Role2 \rightarrow Role1\}$
3. $Time = d, Location = l$
4. $UserRoleAssignment = \{User \rightarrow Role0 \rightarrow Time \rightarrow Location, User \rightarrow Role1 \rightarrow Time \rightarrow Location, User \rightarrow Role2 \rightarrow Time \rightarrow Location\}$

Substituting x and y in W_SSoD_URA predicate with $Role2$ and $Role1$ respectively, we get the violation.

Similar types of analysis reveals that the various forms of SSoD permission role inheritance conflict with the different forms of permission inheritance hierarchy. Conflicts were also detected with the various forms of SSoD user role assignment with different forms of permission inheritance hierarchy. Further, the various forms of DSoD constraints conflict with the different forms of role activation hierarchy. Another source of conflict occurs between role activation and permission when the corresponding location constraints or the temporal constraints do not overlap.

2.3 Conclusion and Future Work

Traditional access control models do not take into account environmental factors before making access decisions. Thus, these models are not quite suitable for pervasive computing applications. Towards this end, we propose a spatio-temporal role based access control model. We identify the entities and relations in traditional RBAC and investigate their relationships with location and time. These relationships necessitate changes in the invariants and the operations of RBAC. The behavior of the new model is formalized using constraints. We investigate the relationships between the different constraints and how they interact with each other.

There still remains some work to be done. We need to investigate how to store location and temporal information in an optimal manner, so that they can be used by the access control enforcement module. Pervasive computing applications are typically represented as workflows. This necessitates our developing a spatio-temporal access control model for workflows. Workflows have additional control-flow and data-flow dependency constraints. It would be interesting to see how these constraints are affected by the spatio-temporal authorization constraints.

Chapter 3

Applying Spatio-Temporal Model to Real-World Applications

The proposed spatio-temporal role-based access control is suitable for various types of application. However, when such a model with numerous features is used for protecting a given application, we must provide assurance that no access control breach occurs. We propose a methodology that describes how we can get assurances that an application is indeed adequately protected. We use a real-world application called the Dengue Decision Support (DDS) system to illustrate our approach. The DDS application is being developed by the Colorado State University in collaboration with the government of Mexico to help state-level public health officials respond to local outbreaks of dengue. Health officials are provided with mobile phones that run this application. They move from location to location gathering statistics about mosquito population which is then uploaded to a central system for further analysis.

In order to formally analyze the authorization policies for the application, it is important to specify the application and its access control requirements in a formal specification language. We chose the Unified Modeling Language (UML) [34] for several reasons. First, it is the de facto modeling language used in the software industry. Second, it is easy to use and understand. Third, it is used together with Object Constraint Language (OCL), which is based on first order predicate logic; this makes specifications in UML amenable to analysis. We show how the existing access control requirements for the DDS can be specified using UML and OCL.

Although formal analysis can be done on UML specifications that are augmented with OCL constraints, there is not much tool support for automated analysis. Towards this end, we advocate the use of Alloy [24] for doing automated analysis. We collaborated with researchers at University of Birmingham, U.K., in the development of a tool called UML2Alloy [1, 2] that automatically transforms UML class diagrams and OCL statements into Alloy models, which can then be verified by the Alloy Analyzer. The analysis demonstrates how well the access control requirements protect the application.

The rest of the chapter is organized as follows. Section 3.1 provides a brief background on how UML models can be transformed into Alloy specifications. Section 3.2 describes the Dengue Decision Support System and its access control requirements. Section 3.3 illustrates the model analysis process in the context of DDS. Section 3.4 concludes the chapter and enumerates future research.

3.1 UML to Alloy Transformation

We propose an approach that will transform UML models with OCL constraints into an Alloy specification. Alloy [22, 23, 24, 57] is a fully declarative first-order logic language designed for modeling and analyzing complex systems. An Alloy model consists of a number of signature and relation declarations. A signature specifies entities used to model the system, and relation declarations specify the dependencies between such entities, allowing the designer to capture complex structures. Alloy is supported by a fully automated constraint solver, called Alloy Analyzer , that analyzes system properties by searching for model instances that violate assertions about them. Alloy Analyzer translates the model into a Boolean expression, and analyzes it using embedded SAT-solvers. The user specifies a scope to the tool, which is an integer number used to bound the domain of model elements. Bounding enables the tool to create finite Boolean formulas for evaluation by the SAT-solver. If Alloy Analyzer produces an instance that violates the assertion (a counterexample), we can infer that the specified property is not satisfied.

There are clear similarities between Alloy and UML languages such as class diagrams and OCL. From a semantic point of view both Alloy and UML can be interpreted by sets of tuples [25, 44]. Alloy is based on first-order logic and is well suited for expressing constraints on object oriented models. Similarly, OCL has extensive constructs for expressing constraints as first order logic formulas. Considering such similarities, model transformation from UML class diagrams and OCL to Alloy seems straightforward. However, UML and Alloy have fundamental differences, which are deeply rooted in their underlying design decisions. For example, Alloy makes no distinction between sets, scalars and relations, while the UML makes a clear distinction between the three. Other examples include that UML supports a number of primitive types, whereas Alloy only supports integers. UML also supports aggregation and composition, but there is no counterpart in Alloy. All of this makes the transformation from UML to Alloy challenging.

Figure 3.1 depicts an outline of our approach. Using the Extended Backus-Naur Form (EBNF) representation of the Alloy grammar [25], we first generate a Meta Object Facility (MOF) compliant [36] metamodel for Alloy. We then select a subset of the class diagrams [33] and OCL [37] metamodels. To conduct the model transformation, a set of transformation rules has been defined. The rules map elements of the metamodels of class diagram and OCL into the elements of the metamodel of Alloy. The rules have been implemented into a prototype tool called UML2Alloy.

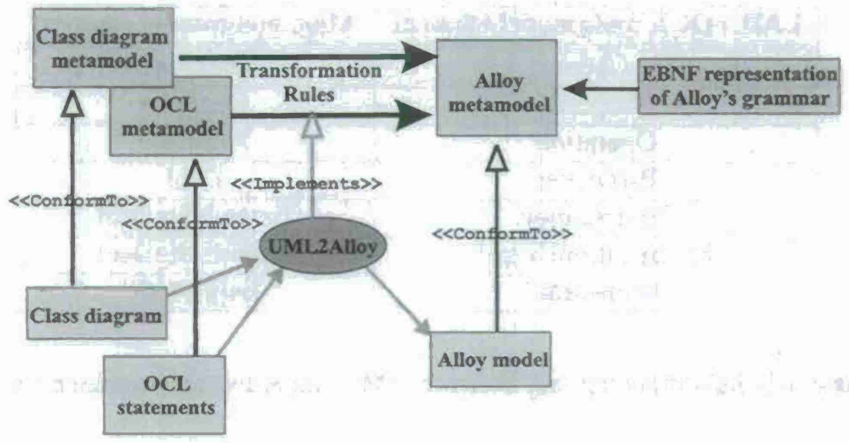


Figure 3.1: Outline of the transformation method

If a UML class diagram, which conforms to the subset of UML we support, is provided as input to UML2Alloy, it automatically generates an Alloy model. For lack of space, we do not show how the EBNF representation of Alloy's grammar is transformed into a MOF compliant metamodel but refer the interested reader to [1]. In addition, the UML and OCL metamodels are not presented here, but can be found in the respective specification documents [33, 37].

Table 3.1 presents a table which provides an informal mapping between the most important elements of the UML and OCL metamodels and Alloy. More specifically a UML *Class* is translated to an Alloy signature declaration (*ExtendsSigDecl*), which defines a *SigId* with the same name. If the class is not a specialization, the Alloy signature is not related to any *SigRef*. Otherwise, it may be related to a *SigRef*, which references the signature it might extend.

A *Property* is translated to a declaration expression (*declExp*), which is used to define a field in an Alloy model. An *Operation* is transformed to a *Predicate* and the *Parameters* of the operation are transformed to declarations (*Decl*). An *Enumeration* [33] is transformed to a signature declaration *SigDecl*, which declares an abstract signature. An *EnumerationLiteral* is transformed to a sub signature. A more complete transformation rules from UML to Alloy and their implementation are explained in our paper [1].

3.2 Dengue Decision Support System

We illustrate our approach using a real-world Dengue Decision Support (DDS) system. DDS helps state-level public health officials respond to local outbreaks of dengue. Response consists of vector control and vector surveillance, namely, spraying (control) and investigating locations where mosquitoes may be breeding and living (surveillance), and if the level of confirmed dengue cases has increased above a prescribed threshold. Public health officials are organized in jurisdictions,

UML+OCL metamodel element	Alloy metamodel element
Class	ExtendsSigDecl
Property	DeclExp
Operation	Predicate
Parameter	Decl
Enumeration	ExtendsSigDecl
EnumerationLiteral	ExtendsSigDecl
Constraint	Expression

Table 3.1: Informal mapping between UML and Alloy metamodel elements

based on population, and multiple jurisdictions are included in a single state. When the threshold is reached, officials at both levels respond. The jurisdiction officer activates vector control and surveillance teams that are local to the jurisdiction, with instructions regarding the specific control and surveillance protocols to follow and the locations where they are to be performed. The state officer releases materials for control to the team, and the local team then performs the controls and surveillance ordered. The jurisdiction and state vector control officials are often located in different buildings, although the vector control team is co-located with the jurisdiction officer. All control materials are located in warehouses elsewhere, and for coordination reasons are controlled by the state officer. Information about specific cases of dengue is retained in what is called an epidemiological study. This data includes information about the patient, the location where the patient lives (the premise), the case, and control and surveillance actions performed at the premise. The patient and case data are considered private information, and are only available to epidemiologists at the jurisdiction and state levels. The vector control team receives premise information along with orders for control and surveillance. However, the team also needs to have names associated with the premises in order to validate the location. The team therefore needs access to some of the patient data for a fixed period of time, in order to perform control and surveillance duties. For lack of space, we omit giving the full specification.

Security Policies

Entities

DDS system consists of the following roles: *State Epidemiologist*, *Jurisdiction Epidemiologist*, *Clinic Epidemiologist*, *Clinician*, *State Vector Control*, *Jurisdiction Vector Control*, and *Local Jurisdiction VC Team*. Tasks user can perform are listed in Table 3.2. Each role can perform their own set of tasks in the designated location and time summarized in Table 3.3.

	Task		Task
1	Read Premise	10	Read VControl
2	Change Premise	11	Change VControl
3	Read Case	12	Read Work Record
4	Change Case	13	Change Work Record
5	Read Patient	14	Read VC Materials
6	Change Patient	15	Change VC Materials
7	Read Patient Names	16	Signal VC Need for DV
8	Read Schedule Work	17	Signal VC Need for DHF
9	Change Schedule Work		

Table 3.2: DDS Tasks List

Role	Tasks	Location Constraint	Time Constraint
State Epi	16	A–State Office	a–Regular Hours
Juris Epi	1, 3	B–Juris Office	a–Regular Hours
	17	B–Juris Office	b–Any Time
Clinic Epi	17	C–Clinic	b–Any Time
Clinician	1, 2, 3, 4, 5, 6	C–Clinic	a–Regular Hours
State VC	11, 15	A–State Office	a–Regular Hours
Juris VC	1, 8, 9, 10, 12, 14	B–Juris Office	a–Regular Hours
Local VC Team	7	B–Juris Office, E–Emergency Location	c–24 Hours Window after signal to begin work
	1, 9, 13	B–Juris Office, D–Field	a–Regular Hours

Table 3.3: DDS Role Constraints

Role Hierarchy

Some roles in the DDS are related using unrestricted permission inheritance hierarchy. Using our model, these relationships can be defined as follow: *State Epi* $\geq$ *Juris Epi*, *Clinic Epi* $\geq$ *Clinician*, and *State VC* $\geq$ *Juris VC*.

Separation of Duty

There are two separation of duty constraints in DDS system. Both are the strong spatial form of static separation of duty. These permissions should not be assigned to the same user at the same time at any location. Note however, unlike traditional separation of duty, these permissions can be assigned to the same user at different times.

1. User should not have permission to change VC protocols at the same time as he has permission to change VC materials.

2. User should not have permission to signal DV at the same time as signal DHF.

These can be represented in STRBAC as follow: $(11,15) \in SSOD\_PRA_l$ and $(16,17) \in SSOD\_PRA_l$.

3.3 Model Analysis

Security analysis begins with abstracting and transforming the security policies of DDS into a UML class diagram and accompanying OCL statements. The class diagram depicts the entities that take part in the model, and defines their attributes related in the access control operations, such as the time and location attribute. OCL statements specify the invariants of the model such as the tasks assigned to role and security constraints that all entities in the model must satisfy. The next step involves using UML2Alloy to automatically transform the class diagram and OCL statements into an Alloy model, which is subsequently analyzed using Alloy Analyzer.

3.3.1 Stage 1: Model Abstraction

We first simplify the original model by removing non-essential elements so that the translation to Alloy produces a model that only contains items necessary to reason about its security properties. For example, we remove the attributes which are not related with the security such as, *gender*, *birthdate*, *ssid* from the *Person* entity since these attributes are not related with the access control model. The resulting UML class diagram is shown in Figure 3.2.

The permission to role assignments are expressed as OCL constraints. The following OCL statements depict the constraints for the permission to role assignment for *Juris Epi* role.

```
context JurisEpi
inv jurisEpiCon : (self.tasks = (Task :: ONE ->
including (Task :: THREE)) and
self.location = Location :: B and
self.timeCon = Time :: a) or
(self.tasks = (Task :: SEVENTEEN -> including
(Task :: SEVENTEEN)) and
self.location = Location :: B and
self.timeCon = Time :: b )
```

The effect of permission inheritance hierarchy and separation of duty can also be expressed in OCL. We omit those details here but refer the interested reader to our paper [53].

3.3.2 Stage 2: Model Transformation

The UML2Alloy tool is used to create an Alloy model from the class diagram and associated OCL specification. When we apply UML2Alloy to the UML class diagram and its OCL specification,

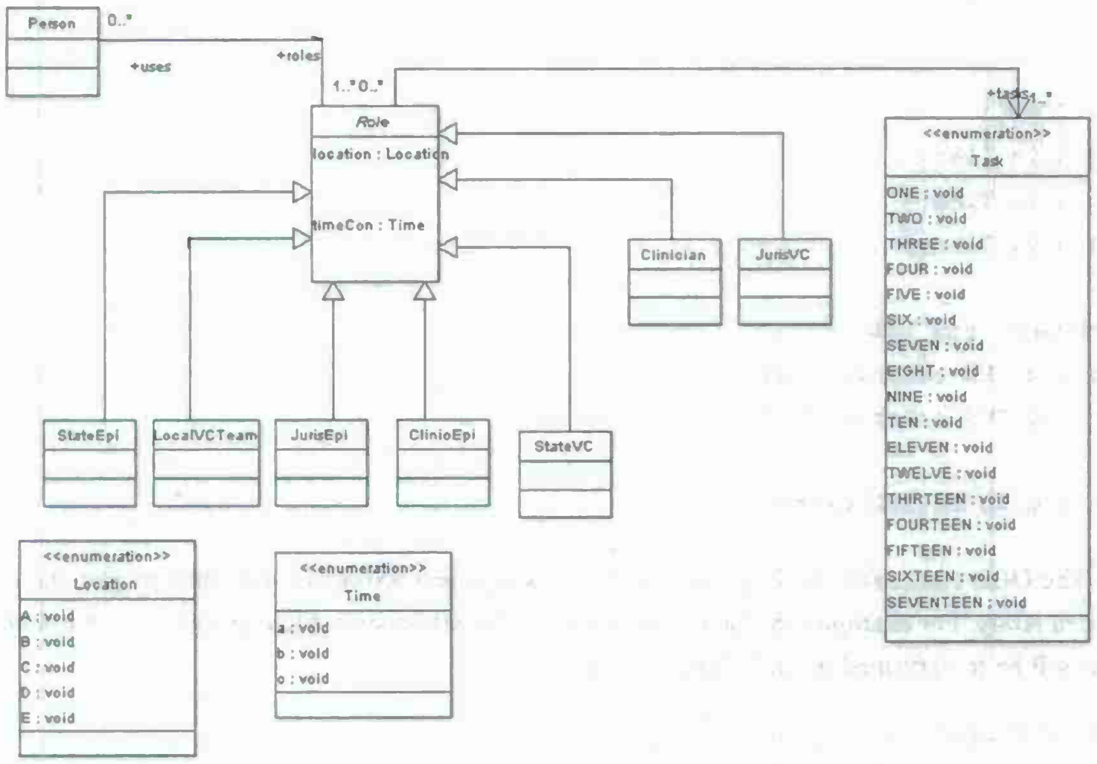


Figure 3.2: UML model for DDS's access control policies

the class diagram will be transformed to the following *signatures* in Alloy corresponding to each class shown in Figure 3.2.

```
abstract  sig Role{
location:one Location,
timeCon:one Time,
tasks:some Task,
uses:set Person}

one sig StateEpi extends Role{}
one sig JurisEpi extends Role{}
....

some sig Person{roles:some Role}

abstract  sig Location{}
one sig A extends Location{}
one sig B extends Location{}
```

```
...

sig Time{}
sig a in Time{}
sig b in Time{}
sig c in Time{}

abstract  sig Task{}
one sig ONE extends Task{}
one sig TWO extends Task{}
...
one sig SEVENTEEN extends Task{}
```

The OCL constraint for the permission role assignment will be transformed to *fact* and *predicate* in Alloy. For example, the OCL constraint for the permission role assignment of the *Juris Epi* role will be transformed to the following Alloy code.

```
fact JurisEpi_jurisEpiCon_fact{
all self: JurisEpi | JurisEpi_jurisEpiCon[self]}

pred JurisEpi_jurisEpiCon[self: JurisEpi]{
((self.tasks = ONE+THREE) && (self.location = B) &&
(self.timeCon = a)) || ((self.tasks = SEVENTEEN) &&
(self.location = B) && (self.timeCon in Time))}
```

The effect of role hierarchy represented in the OCL constraint will also be transformed to *fact* and *predicate* in Alloy. The OCL constraint for the separation of duty constraint will be transformed to *predicate* in Alloy. Our paper [53] lists all the detailed specifications.

3.3.3 Stage 3: Model Analysis

Alloy assertions must be formulated prior to analysis by Alloy Analyzer. Assertions are statements that capture properties we wish to verify. Alloy Analyzer automatically checks such assertions and, if they fail, produces a counterexample. We have checked several assertions regarding the security properties of the example system. For example, it is crucial to ensure that no user can change VC protocols (task 11) at the same time as he has permission to change VC materials (task 15). To verify this, we create the following assertion:

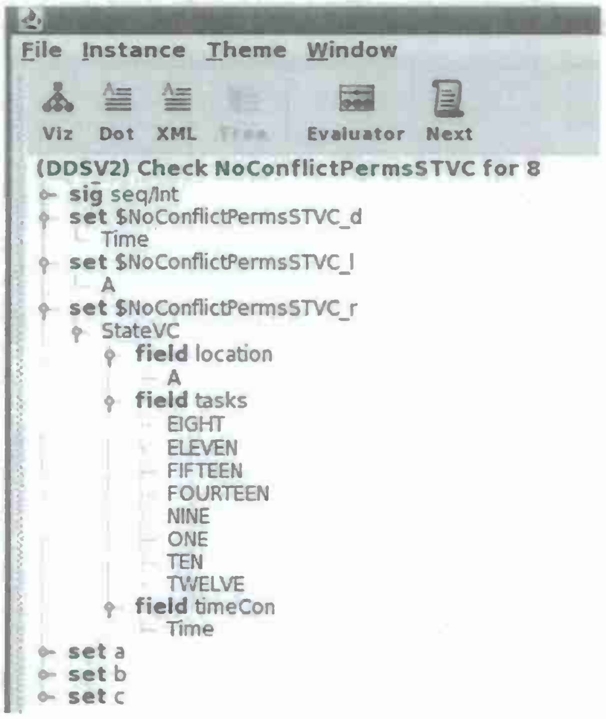
```
assert NoConflictPermsSTVCAssigned{
```

```
all r: Person.roles, d: Time, l: Location|
((ELEVEN in r.tasks) && (d in r.timeCon) &&
(l in r.location)) =>
((FIFTEEN !in r.tasks) && (d in r.timeCon) &&
(l in r.location))}
```

The assertion produced no counterexample, meaning that it is valid for the given scope, which in this case was 8. We also checked whether the SoD for role permission assignment is maintained.

```
assert NoConflictPermsSTVC{
all r: StateVC, d: Time, l: Location|
((ELEVEN in r.tasks) && (d in r.timeCon) &&
(l in r.location)) =>
((FIFTEEN !in r.tasks) && (d in r.timeCon) &&
(l in r.location))}
```

We chose a value of 8 for the scope of this analysis as well. However, this time the analyzer produced counterexample, which means these conflict permissions can be assigned to the same role. The counterexample is shown in Figure 3.3.



```
(DDSV2) Check NoConflictPermsSTVC for 8
sig seq/Int
set $NoConflictPermsSTVC_d
  Time
set $NoConflictPermsSTVC_l
  A
set $NoConflictPermsSTVC_r
  StateVC
    field location
      A
    field tasks
      EIGHT
      ELEVEN
      FIFTEEN
      FOURTEEN
      NINE
      ONE
      TEN
      TWELVE
    field timeCon
      Time
set a
set b
set c
```

Figure 3.3: Counterexample for assertion *NoConflictPermsSTVC*

3.4 Conclusion and Future Work

Our spatio-temporal access control model is well suited for securing real-world pervasive computing applications. However, due to the complexity of the application and the access control model, we need assurance that the application is indeed adequately protected. We use UML together with OCL for specifying the application and its access control requirements. Since UML does not have much automated tool support, we convert the UML model into Alloy and verify the resulting model automatically. In this chapter, we showed how the specification and verification of a typical application security policies can be effected in our framework.

The applicability of SAT-solvers (such as the one in Alloy) for the purpose of analysis is limited by the size of the model that can be verified. Consequently, we are investigating how to further abstract the model resulting in the construction of smaller SAT formulae that can be efficiently verified. This, together with new research for improving SAT-solver technology, will alleviate the limitation mentioned above.

Chapter 4

Graph-Theoretic Representation of Spatio-Temporal Model

In the previous chapters, we outlined the new spatio-temporal role-based access control model that we have developed as part of this project. The model is specified in terms of constraints that support the various features of the model. It is important that the model be properly analyzed which is a non-trivial problem. In this chapter, we refine our original spatio-temporal role-based access control model into three simpler models so that their semantics can be expressed expressed in terms of graph theory. The use of graph-theory offers several advantages. It allows one to visualize the relationships and interactions among the different components of the model. Using the directed graph representation, the interaction and relationship between components in the model becomes more clear and expressive. It also allows one to readily detect the presence of inconsistencies using graph theoretic algorithms. These simple spatio-temporal access control models are more easily used in real world applications.

The simpler models also serve an important purpose. Pervasive computing applications, in general, are dynamic in nature. This means that while an application is executing, the entities requiring access or the resources needing protection may change. In the face of such dynamism, it is essential to ensure that access control breaches do not occur. Since the required analysis to verify the satisfaction of security properties must also be done in real-time, it is important to minimize the verification time. The graph-theoretic approach allows techniques for incremental analysis with good time complexity results. For example, to detect SoD violations in a dynamic graph, we need to find whether the nodes connected by SoD constraints have a common predecessor. Applying a naïve algorithm based on Depth First Search, requires $O(kE)$ time for each change applied to the graph, where k is the number of SoD constraints and E is the number of edges. We have been able to improve upon this result significantly by proposing a new common predecessor detecting algorithm in a dynamic graph.

The rest of the chapter is organized as follows. Section 4.1 presents our spatio-temporal role-based access control model using graph theoretic notations. Section 4.2 focuses on the dynamic aspects of the model and how we can ensure absence of access control breaches in the face of such changes. Section 4.3 illustrates our ideas by using an example application. Section 4.4 concludes the chapter.

4.1 STARBACD: The Refined Spatio-Temporal Model

We begin by giving a graph-theoretic formulation for our spatio-temporal role-based access control model that supports role hierarchy. The set of vertices $V = U \cup R \cup P \cup O$ correspond to the RBAC entities: Users (U), Roles (R), Permissions (P), and Objects (O). Our model assumes the existence of the following relationships of RBAC that constitute the set of edges $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$ where

- User-Role Assignment (UA) = $U \times R$
- Permission-Role Assignment (PA) = $R \times P$
- Permission-Object Assignment (PO) = $P \times O$
- Role Hierarchy (RH) = $R \times R \times \{a, u\}$, which can be categorized to:
  - the activation hierarchy (RH_a) = $\{(r, r') : (r, r', a) \in RH\}$, and
  - the permission usage hierarchy (RH_u) = $\{(r, r') : (r, r', u) \in RH\}$

We define the notion of activation path, usage path and access path in a manner inspired by Chen and Crampton [8]. An *activation path* (or *act-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$ such that $(v_1, v_2) \in UA$ and $(v_{i-1}, v_i) \in RH_a$ for $i = 3, \ldots, n$. A *usage path* (or *u-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in RH_u$ for $i = 1, \ldots, n-2$, and $(v_{n-1}, v_n) \in PA$. An *access path* (or *acs-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$, such that (v_1, v_i) is an act-path, (v_i, v_{n-1}) is an u-path, and $(v_{n-1}, v_n) \in PO$.

We assume the existence of a spatio-temporal domain $\mathcal{D}$. We develop three refined models, namely, the standard model (STARBACD$^=$), the strong model (STARBACD$^+$), and the weak model (STARBACD$^-$). The models differ with respect to the spatio-temporal constraints that must be satisfied by the entities for the authorization to be successful. The strong model imposes the most number of constraints and is suitable for military applications. The weak model imposes the least number of constraints. It is intended primarily for emergency situations where we need to make rapid decisions yet ensuring that minimum security requirements are not violated. The details of all three models appear in our paper [54]. We present the highlights of the strong model only in this chapter.

The strong model is used when the individual entities (users, roles, permissions, objects) and the different relationships must satisfy the spatio-temporal constraints. Each entity is associated with spatio-temporal points that indicate where the entity can be activated. For example, the spatio-temporal points associated with a role specify when and where the role can be activated. Similarly, the spatio-temporal points associated with a relation indicate when the relationship can be activated. To illustrate, consider the relation $(r,p) \in PA$. In this case, we not only have to take into account the spatio-temporal points at which the role r can be activated in a session and the points at which the permission p can be invoked, but also we must consider the spatio-temporal points when r can invoke p.

The spatio-temporal constraints in the *strong STARBACD model* (or STARBACD$^+$) are described using two functions λ and μ which are defined below. $\lambda : V \to 2^{\mathcal{D}}$. For $v \in V$, $\lambda(v) \subseteq \mathcal{D}$ denotes the set of points in space-time at which v can be invoked.

- if $u \in U$, then $\lambda(u)$ denotes the set of points in space-time at which u may create a session;
- if $r \in R$, then $\lambda(r)$ denotes the set of points in space-time at which r may be activated in a session;
- if $p \in P$, then $\lambda(p)$ denotes the set of points in space-time at which p may be granted;
- if $o \in O$, then $\lambda(o)$ denotes the set of points in space-time at which o may be accessible.

$\mu : E \to 2^{\mathcal{D}}$. For $e = (v,v') \in E$, $\mu(v,v')$ denotes the set of points in space-time at which the association between v and v' is enabled.

- if $(u,r) \in UA$, then $\mu(u,r)$ denotes the set of points in space-time at which u is assigned to r;
- if $(r',r) \in RH_a$, then $\mu(r',r)$ denotes the set of points in space-time at which r' is senior to r in the activation hierarchy;
- if $(r',r) \in RH_u$, then $\mu(r',r)$ denotes the set of points in space-time at which r' is senior to r in the permission usage hierarchy;
- if $(r,p) \in PA$, then $\mu(r,p)$ denotes the set of points in space-time at which p is assigned to r.
- if $(p,o) \in PO$, then $\mu(p,o)$ denotes the set of points in space-time at which o is assigned to p.

Given a path $v_1,\ldots,v_n$ in the labeled graph $G = (V,E,\lambda,\mu)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, we write $\hat{\mu}(v_1,\ldots,v_n) = \hat{\mu}(v_1,v_n) \subseteq \mathcal{D}$ to denote $\bigcap_{i=1}^{n-1} \mu(v_i,v_{i+1})$. The semantics imply that an edge can only be enabled if both endpoints are enabled. Hence, $\hat{\mu}(v_1,v_n)$ is the set of points at which every vertex and every edge in the path is enabled.

Authorization in STARBACD$^+$:
- a user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an act-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \hat{\mu}(v,v')$;
- a role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists an u-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \hat{\mu}(v,v')$;
- a user $v \in U$ is authorized for permission $v' \in P$ with respect to object $v'' \in O$ at point $d \in \mathcal{D}$ if and only if there exists an acs-path $v = v_1, v_2, \ldots, v_i, \ldots, v_{n-1} = v', v_n = v''$ such

that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_i, \ldots, v_{n-1}$ is an u-path, $(v_{n-1}, v_n) \in PO$ and $d \in \hat{\mu}(v, v'')$;

Our model also supports separation of duty (SoD) constraints. SoD prevents the occurrence of fraud arising out of conflicts of interests in organizations [46]. Separation of Duty (SoD) comes in two varieties. First one ensures that no user can be assigned to two conflicting roles. Second one guarantees that no role can be assigned two conflicting permissions. We denote these two types of SoD by using SD^R and SD^P edges, respectively. Since SoD is a symmetric relationship, the SD^R and SD^P edges are bi-directional.

The strong model supporting SoD constraints is defined over the labeled graph $G = (V, E, \lambda, \mu)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u \cup SD^R \cup SD^P$ and $V = U \cup R \cup P \cup O$. The strong model allows specification of weaker forms of SoD constraints than those supported by the traditional RBAC. Specifically, it allows one to specify the spatio-temporal points at which the SoD constraints are valid.

SoD Constraints for STARBACD$^+$

User-Role Assignment: if $(r, r') \in SD^R$ then there are no two edges (u, r) and (u, r'), corresponding to some user u, where $\mu(u, r) \cap \mu(u, r') \cap \mu(r, r') \neq \emptyset$

Permission-Role Assignment: if $(p, p') \in SD^P$ then there are no two u-paths $r = v_1, v_2, \ldots, v_n = p$ and $r = v'_1, v'_2, \ldots, v'_n = p'$ where $\hat{\mu}(v_1, v_n) \cap \hat{\mu}(v'_1, v'_n) \cap \mu(p, p') \neq \emptyset$

Pervasive computing applications require that our model support delegation. This is because many situations require the temporary transfer or granting of access rights belonging to a user/role to another user/role in order to accomplish a given task. For example, a doctor may delegate some of his privilege to the nurse while he is temporarily unavailable. The entity that transfers or grants its privileges temporarily to another entity is referred to as the delegator and the entity who receives the privilege is known as the delegatee. The delegator (delegatee) can be either an user or a role. Thus, we may have four types of delegations: *user to user* (U2U), *user to role* (U2R), *role to role* (R2R), and *role to user* (R2U). When a user is the delegator, he can delegate a subset of permissions that he possesses by virtue of being assigned to different roles. When a role is the delegator, he can delegate either a set of permissions or he can delegate the entire role. We can therefore classify delegation on the basis of role delegation or permission delegation. In the graphical representation of STARBACD, we define a function $\nu : (U \cup R) \times (R \cup P) \rightarrow (U \cup R)$ that maps the delegation to the delegator. The user to user role delegation is formalized as follows: $(Delegate^R_{U2U}) = U \times R$, $\nu(u, r') = u'$ denotes the delegator who is a user authorized for role r'. The other types of delegation can be formalized similarly.

Delegation in the Strong Model STARBACD$^+$

The strong model supporting delegation is defined over the labeled graph $G = (V, E, \lambda, \mu)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u \cup DG$ and DG is the set of all delegation edges and $V =$

$U \cup R \cup P \cup O$. We specify a sample delegation constraint as follows: If $(u,r') \in Delegate^{R}_{U2U}$ and $\nu(u,r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\hat{\mu}(v_1, v_n) \cap \mu(u,r') \neq \emptyset$ This says that when a user u' delegates role r' to user u, then the delegation is possible only if the spatio-temporal points for activating user u''s role r' overlap with those in which the delegation is valid. For lack of space, we do not discuss all the other forms of delegation constraints, but refer the reader to our paper [54].

4.2 Dynamic Model

Pervasive computing applications are dynamic in nature–the accessing entities may change, resources requiring protection may be created or modified, and an entity's access to resources may change during the course of the application. Such changes may result in unreachable or isolated entities (such as, a normally authorized user being denied access because the user-role-permission assignment has been removed), or the violation of separation of duty constraints. We need to analyze the model to detect such problems. The following changes are possible in our model.

1. **Entity and Relationship Removal** The following entities can be removed: user, role, permission, or object. Note that, this removal must be accompanied by deleting the relationships associated with these entities.
2. **Relationship Removal** The following relationships can be removed: User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, or Permission-Object Assignment. This type of change can also cause an entity to become isolated.
3. **Relationship Creation** A new relationship can be created between existing entities. The relationship may be User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, Permission-Object Assignment, SoD, or Delegation. Creation of a new relationship may result in separation of duty violation.
4. **Entity and Relationship Creation** A new entity together with its corresponding new relationship can be created. The entity may be user, role, permission, or object. The relationship may be User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, Permission-Object Assignment, SoD, or Delegation depending on the type of entity being created. This type of change can cause the SoD constraints violation.
5. **Updating Spatio-Temporal Constraints** The spatio-temporal constraints assigned to entities or relations can be changed. The entity may be user, role, permission, or object. The relationship may be User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, Permission-Object Assignment, SoD, or Delegation. This type of change can cause either the infeasible path violation or SoD constraints

violation.

4.2.1 Algorithm for Detecting Isolated Entities

Preliminaries

We define an isolated entity as one which is unreachable and therefore cannot be used. The isolated entity can be determined by considering the *in-degree* and *out-degree* of each vertex. The *in-degree* and *out-degree* of the vertex defined with respect to STARBACD$^+$ model are given below.

In-degree In the labeled graph $G = (V, E, \lambda, \mu)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, *in-degree* of a vertex v is the cardinality of the set $\{(v', v) | ((v', v) \in E) \wedge (\lambda(v') \cap \lambda(v) \cap \mu(v', v) \neq \emptyset)\}$

Out-degree In the labeled graph $G = (V, E, \lambda, \mu)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, *out-degree* of a vertex v is the cardinality of the set $\{(v, v') | ((v, v') \in E) \wedge (\lambda(v) \cap \lambda(v') \cap \mu(v, v') \neq \emptyset)\}$

Note that, we do not consider the separation of duty or the delegation edges since the modifications to these edges do not change the isolated entities.

The Detection Algorithm

The different types of isolated entities are detected as follows:

User For $v \in U$, v is the isolated entity iff *out-degree*$(v) = 0$

Role and Permission For $v \in R \cup P$, v is the isolated entity iff (*in-degree*$(v) = 0$) $\vee$ (*out-degree*$(v) = 0$)

Object For $v \in O$, v is the isolated entity iff *in-degree*$(v) = 0$

To get the *in-degree* and *out-degree*, we have to count the number of edges connected to each vertex. This can be done in $O(VE)$ time. However, we can improve this by recording the *in-degree* and *out-degree* of each vertex. Each time the vertex or the edge is added to or removed from the graph, we update the *in-degree* and *out-degree* of the related vertices. Since we do not allow the existence of multiple edges between each pair of vertices, this update process can be done in $O(V)$. After we have such values recorded for every vertex, the detection can be done in $O(V)$.

4.2.2 Algorithm for Detecting Infeasible Paths

Preliminaries

In STARBACD model, a user u is authorized for permission p through role r with respect to object o iff there exists a valid *acs-path* which contains u, r, p, and o. We define an infeasible path as an invalid *acs-path* i.e. an *acs-path* which cannot grant the authorization of any permission to user.

The Detection Algorithm

To detect the infeasible path, we assume that we store all source vertices in a list. Each member in the list maintain its own depth-first search (DFS) tree. To generate these trees, we perform DFS from each source. While performing the DFS, we check if there is any spatio-temporal conflicts between the nodes or edges. If there is any conflict, then there exists an infeasible path. This step could be done in $O(VE)$. After the process we will have set of the initial DFS trees which consists of feasible paths. Next for each update operation of the graph, we ensure that the following conditions are satisfied:

- Only *user* vertices can be the root of each subtree.
- Only *object* vertices can be the leaf node of each subtree.

For each update operation of the graph, we proceed as discussed here. If any new entity v and its corresponding relationship have been added to the initial graph, we consider the following:

- If v is a new source, we perform DFS from v to create all of its *acs-path*s. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we add v to the source list and maintain its pointers to its immediate successors. If not, then this v will create an infeasible path. This step can be done in $O(E)$ time.
- If v is a new intermediate vertex, we perform DFS from each source. While performing the DFS, we check whether all spatio-temporal constraints are satisfied. If so, we create pointer from v's immediate predecessors to v, and from v to its immediate successors. If not, then this v will create an infeasible path. This step can be done in $O(VE)$ time.
- If v is a new sink, we perform reverse DFS from v. While performing the reverse DFS, we check whether the spatio-temporal constraints between v and its predecessors are satisfied. If so, we create pointer from its immediate predecessors to v. If not, then this v will create an infeasible path. This step can be done in $O(E)$ time.

If any existing spatio-temporal constraint has been updated in the initial graph, we consider the following:

- If the update is done on $\lambda(v)$, where v is a source, we perform DFS from v to each of its *acs-path*. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we update $\lambda(v)$ to the new one. If not, then this update will create an infeasible path.This step can be done in $O(E)$ time.
- If the update is done on $\mu(v,v')$, where v is a source, we perform DFS from v to each of its $acs-path$ which contains v'. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we update

$\mu(v,v')$ to the new one. If not, then this update will create an infeasible path.This step can be done in $O(E)$ time.

- If the update is done on $\lambda(v)$ or $\mu(v,v')$, where v is an intermediate vertex, we perform DFS from each source. While performing the DFS, we check whether all spatio-temporal constraints are satisfied. If so, we update $\lambda(v)$ or $\mu(v,v')$ to the new one. If not, then this update will create an infeasible path. This step can be done in $O(VE)$ time.
- If the update is done on $\lambda(v)$, where v is a sink, we perform reverse DFS from v. While performing the reverse DFS, we check whether the spatio-temporal constraints between v and its predecessors are satisfied. If so, we update $\lambda(v)$ to the new one. If not, then this v will create an infeasible path. This step can be done in $O(E)$ time.
- If the update is done on $\mu(v,v')$, where v' is a sink, we perform DFS from v' to each of its *acs-path* which contains v. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we update $\mu(v,v')$ to the new one. If not, then this update will create an infeasible path. This step can be done in $O(E)$ time.

4.2.3 Algorithm for Detecting SoD Violations

Preliminaries

In STARBACD model, SoD can be violated in one of two ways. First, if $(r_1,r_2) \in SD^R$, and there exists *acs-paths* from u_1 to r_1 and u_1 to r_2. Or, if $(p_1,p_2) \in SD^P$, and there exists *u-paths* from r_1 to p_1 and r_1 to p_2.

The Detection Algorithm

Consider the dynamic case where edges can be added and deleted from the graph. The naïve algorithm can be done by performing the reverse DFS on each $(v,v') \in SD^R \cup SD^P$ of the modified graph to find the common predecessor. This could be done in $O(k|E|)$ time. We can apply the same algorithm for the case where the spatio-temporal constraint is updated in the graph too.

Our algorithm which will be proposed next is a special case of the algorithm to find the common predecessors in a Directed Acyclic Graph (DAG). In our algorithm, each entity except a user will maintain a list of users authorized for it by performing the DFS from each user. Only users satisfying the spatio-temporal constraints will be added to the list. To determine whether the SoD $(v,v') \in SD^P \cup SD^R$ is violated, we compare whether $u \in U$ is in the authorized users list of both v and v', and $\lambda(u) \cap \mu(v,v') \neq \emptyset$. If this evaluates to true, then there exists an SoD violation. Since the size of each list cannot exceed the number of user vertices, the evaluation time is $O(|U|)$. Let k be number of SoD edges. The detection time for the static case where no adding or removing of edges is allowed, is equal to $O(k|U|)$. To label all vertices it takes $O(|E||U|)$ time, and yields the total running time in the static graph equal to $O((k+|E|)|U|)$. However, in the case where all edge modifications are of same type, i.e., only either adding edges or deleting edges are allowed, we can improve the running time by applying the following algorithm:

- When only adding edges is allowed, each time a new edge is added, we update only the label list of vertices belonging to the graph portion that have not been reached before by using the Incremental-DFS described in our paper [54]. All updates take $O(|E||U|)$ time, and detecting whether the SoD is violated take $O(|U|)$ per SoD edge. This yields the total processing time equal to $O((k+|E|)|U|)$.
- When only removing edges is allowed, we update only the label list of vertices that becomes unreachable by some user u after the edge removal. Using our proposed algorithm [54], the removal of an edge takes $O(|E|log|V|)$ time for relabeling for each user vertex, and detecting whether the SoD is violated take $O(|U|)$ per SoD edge. This yields the total processing time equal to $O((k+|E|log|V|)|U|)$.

For the detail on graph specification updating algorithm and proof of correctness, we refer to our paper [54].

4.3 Military Example

We describe a military application where the STARBACD$^+$ can be applied. Let us assume that in the battlefield, each troop consists of military staff with the following responsibilities: The *Intelligent Officer* is responsible for the process of acquiring enemy information, interpreting it and then sending it to the *Soldier* in his troop. The *Clinical Officer* is in charge of monitoring the health information of his troop, evaluating the information to check whether the trooper's life is in danger, and sending the SOS signal to the commander to get the proper help. The list of entities and the spatio-temporal relationships are shown in Tables 4.1 and 4.2 respectively.

The graph-theoretic representation is shown in Figure 4.1(a). We will only describe parts of this configuration. User *Alex* (u_1) can create session at any time and at any place as per Row 1 of Table 4.1. He is assigned the role of *Intelligence Officer* (r_1) which can be activated at any place at any time. During this time and at this location, he has permission (p_1) to access the *Surveillance Sensor Information* (o_1). Since *Intelligence Officer* is senior to *Soldier* role in the permission usage hierarchy, he can also get the permission to maneuver the *Tank*. However, this permission is allowed only when the hierarchy is enabled on the battlefield. During the war, Alex gets injured and cannot pursue his mission. So, his role must be delegated to Charlie until he fully recovers. This new graphical representation is shown in Figure 4.1(b) where the delegation edge is represented by the dashed arrow. However, this delegation should not be allowed because our algorithm detects a violation of separation of duty constraint in the presence of this delegation.

NAME	DESCRIPTION	SPATIO-TEMPORAL DOMAIN (λ)
u_1	Alex	$[Universe, Always]$
u_2	Ben	$[Universe, Always]$
u_3	Charlie	$[Universe, Always]$
r_1	Intelligence Officer	$[Universe, Always]$
r_2	Soldier	$[Field, Always]$
r_3	Clinical Officer	$[Universe, Always]$
p_1	Access Surveillance Sensor	$[Universe, Always]$
p_2	Maneuver the Vehicle	$[Field, Always]$
p_3	Access Vital Sensor	$[Universe, Always]$
o_1	Surveillance Sensor Information	$[Universe, Always]$
o_2	Tank	$[Field, Always]$
o_3	Health Information	$[Universe, Always]$

Table 4.1: STARBACD entities for the military example

NAME	DESCRIPTION	SPATIO-TEMPORAL DOMAIN (μ)
(u_1, r_1)	User-Role Assignment	$[Universe, Always]$
(u_2, r_2)	User-Role Assignment	$[Field, Always]$
(u_3, r_3)	User-Role Assignment	$[Universe, Always]$
(r_1, r_2)	Permission Usage Hierarchy	$[Field, Always]$
(r_1, p_1)	Permission-Role Assignment	$[Universe, Always]$
(r_2, p_2)	Permission-Role Assignment	$[Field, Always]$
(r_3, p_3)	Permission-Role Assignment	$[Universe, Always]$
(p_2, p_3)	Separation of Duties	$[Universe, Always]$
(p_3, p_2)	Separation of Duties	$[Universe, Always]$
(p_1, o_1)	Permission-Object Assignment	$[Universe, Always]$
(p_2, o_2)	Permission-Object Assignment	$[Field, Always]$
(p_3, o_3)	Permission-Object Assignment	$[Universe, Always]$

Table 4.2: STARBACD relationships and constraints for the military example

4.4 Conclusion and Future Work

We present a graph-theoretic representation of our spatio-temporal role-based access control model that allows one to visualize and reason about spatio-temporal access control. The dynamism inherent in pervasive computing applications may cause the access control configuration to change while the application is executing. Towards this end, we show how to perform incremental analysis to give assurance that security breaches do not occur as a result of changing the access control configuration. Our analysis makes clever use of data structures and achieves good time complexity results.

Pervasive computing applications will typically be modeled as workflows. In future, we plan to extend our graph-theoretic formalism to represent the access control configuration of workflows.

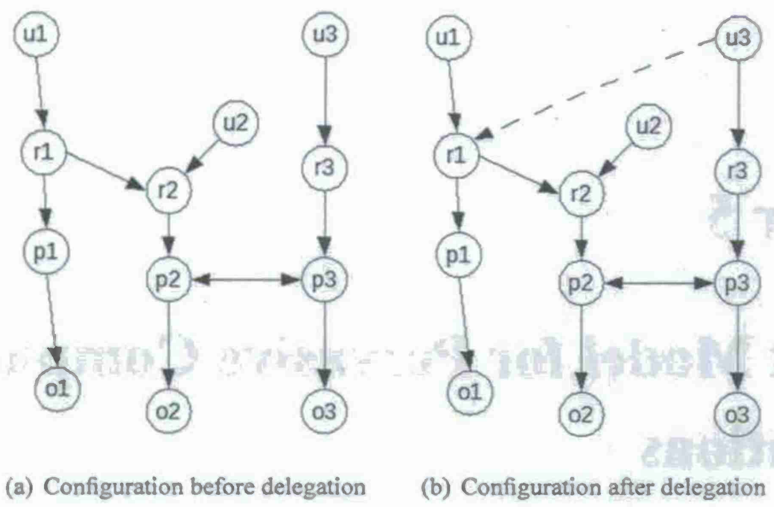

u1
u3
u2
r1
r2
r3
p1
p2
p3
o1
o2
o3
(a) Configuration before delegation
(b) Configuration after delegation

Figure 4.1: Access control configurations for the military example

We also plan to investigate the interaction of workflow and authorization constraints where the access control model is updated during workflow execution.

Chapter 5

A Trust Model for Pervasive Computing Applications

Traditional security policies and mechanisms assume a binary notion of trust – either an entity is trusted completely or not at all. However, such a simplistic notion of trust is not suitable for pervasive computing applications where there are interactions among different entities, not all of which are equally trustworthy. The reason is that these binary models of trust fail to correctly assess trust levels of groups in which some of the entities are trusted while others are not. The nature of interactions, often times, depends on the trust relationships between the entities. Thus, it is important to formalize and capture the trust relationships which will allow us to compare them and compose them to make decisions. Moreover, since pervasive computing applications are dynamic and involves interacting with unknown entities, the trust model should be able to represent and argue about uncertainty. The trust model should also be interoperable as pervasive computing applications often span multiple organizations.

In this chapter, we present the highlights of a new trust model that we propose for pervasive computing applications. We begin by defining trust as a relationship between a truster and a trustee with respect to a given context. We identify the factors on which trust depends and show how to assess these factors and compute the value of trust relationship. Subsequently, we formalize the notion of context that allows us to compare trustworthiness across different domains and also enables one to extrapolate trust in the absence of information in a given context.

The rest of the chapter is organized as follows. Section 5.1 presents an overview of our trust model. Section 5.2 formalizes the relationship among different contexts which is needed to make the trust model interoperable and to reason about trust in the absence of information in a given context. Section 5.3 concludes the chapter with some references to future work.

5.1 Overview of Trust Model

Trust is a relationship between two entities, a *truster* A and a trustee B, with respect to some context c. The trust relationship between a truster and a trustee is never absolute. A truster trusts a trustee with respect to specific capabilities, such as providing a service or keeping a secret. This represents our notion of trust context.

We start by representing the trust relationship $(A \xrightarrow{c} B)_t$ as a 3×3 matrix. The rows of the matrix correspond to the three parameters, namely, *experience*, *knowledge*, and *recommendation*, on which trust depends. (The formal definitions of these parameters and methods for evaluating them are given later.) We use Jøsang's *opinion model*[26] to represent each of these parameters. Each parameter is a (b,d,u) triple, where b means belief, d specifies disbelief, and u signifies uncertainty about the parameter to evaluate the trust. These three terms constitute the columns of the trust matrix.

The three parameters may not have equal importance for evaluating trust. The *trust policy vector* specifies the normalization factor that gives the relative weight of each parameter. Applying the normalization factor to the trust relationship gives a *normalized trust relationship*. The normalized trust relationship between truster A and trustee B pertaining to context c at time t is formally denoted as $(A \xrightarrow{c} B)_t^N$. It specifies A's *normalized* trust on B at a given time t for a particular context c. This normalized trust is represented as a single triple $({}_A\hat{b}_B^c,\ {}_A\hat{d}_B^c,\ {}_A\hat{u}_B^c)$.

Trust is evaluated on the basis of three factors, namely, experience, knowledge, and recommendations. In the following subsections, we briefly describe how each of these factors are computed.

5.1.1 Evaluating Experience

Definition 1 The *experience* of a truster about a trustee is defined as the measure of the cumulative effect of a number of events that were encountered by the truster with respect to the trustee in a particular context and over a specified period of time.

We model experience in terms of the number of events encountered by a truster, A, regarding a trustee, B in the context c within a specified period of time $[t_0, t_n]$. We assume that A has a record of the events since time t_0. An event can be positive or negative or neutral. Positive events contribute towards increasing the belief component of experience. Negative events increase the disbelief component of experience. Neutral events increase both belief and disbelief components equally. No experience contributes towards the uncertainty component of experience. In the following, we describe how to calculate the experience that a truster A has about trustee B with respect to context c. This is formally denoted as ${}_AE_B^c = (b_E,\ d_E,\ u_E)$ where b_E, d_E, u_E represent belief, disbelief and uncertainty components respectively with respect to the experience that A has towards B.

We use the temporal notation $[t_i, t_j]$ for describing a time interval where $t_i \preceq t_j$. The time interval $[t_i, t_j]$ describes the set of consecutive time instances where t_i is the first instance and t_j is the last one. We denote the time period of interest as $[t_0, t_n]$. This is divided into a set of n sub-intervals $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{n-1}, t_n]$. The intervals overlap at the boundary points only. That is, $\forall i, j, k, l \in \mathbb{N}$, where i, j, k, l are all distinct, $[t_i, t_j] \cap [t_k, t_l] = \emptyset$. Also, $\forall i, j, k \in \mathbb{N}$, where i, j, k are all distinct, $[t_i, t_j] \cap [t_j, t_k] = \{t_j\}$. That is, all instances, except t_0 and t_n, that occur at the boundary of an interval is a part of two intervals. We refer to the interval $[t_{k-1}, t_k]$ as the k^{th} interval where $0 \leq k \leq n-1$.

We assume that events occur at time instances. The function ET, referred to as the event-occurrence-time function, returns the time instance t_j at which a given event e_k occurred. Formally, $ET(e_k) = t_j$. Moreover, if $ET(e_k) = t_j$ and $t_j \in [t_i, t_k]$ and $j \neq i \wedge j \neq k$, then e_k is said to occur in the interval $[t_i, t_k]$. For two consecutive intervals $[t_i, t_j]$ and $[t_j, t_k]$ if $ET(e_l) = t_j$ then we assume e_k occurs in the interval $[t_i, t_j]$.

Let the experience acquired at interval i, $1 \leq i \leq n-1$, be represented as (b_i, d_i, u_i) where b_i, d_i, u_i denotes belief, disbelief, and uncertainty respectively. When no event occurs during some particular time interval i, this corresponds to the fact that $u_i = 1$ and $b_i = d_i = 0$. The next case is when events occur at the interval i. Let P_i denote the set of all positive events, Q_i denote the set of all negative events, and N_i denote the set of all neutral events that occur in the interval i. Each positive event increases b_i, each negative event increases d_i, and each neutral event increase both b_i and d_i. The values for b_i, d_i and u_i are computed as follows. $b_i = \frac{|P_i| + \frac{|N_i|}{2}}{|P_i| + |Q_i| + |N_i|}$, $d_i = \frac{|Q_i| + \frac{|N_i|}{2}}{|P_i| + |Q_i| + |N_i|}$, and $u_i = 0$. The intuition is that each positive event contributes to the belief component by $\frac{1}{|P_i| + |Q_i| + |N_i|}$. Similarly, each negative event contributes to the disbelief component by $\frac{1}{|P_i| + |Q_i| + |N_i|}$. Each neutral event contributes equally to both belief and disbelief component by $\frac{0.5 * |N_i|}{|P_i| + |Q_i| + |N_i|}$. Moreover, since events have occurred in the interval, the uncertainty component is 0.

Note that, in real world, events occurring in the distant past has less effect than those that have recently occurred. More importance must be given to recent events than past ones. To accommodate this in our model, we assign a *non-negative* weight w_i to the i^{th} interval such that $w_i > w_j$ whenever $j < i$, $i, j \in \mathbb{N}$. We use the formula $w_i = \frac{i}{S}$ $\forall i = 1, 2, \ldots, n$ where $S = \frac{n(n+1)}{2}$ to evaluate weights of the intervals, satisfying the above condition.

The experience of A about B in context c is expressed as, ${}_AE_B^c = (b_E,\ d_E,\ u_E)$. The values of b_E, d_E, and u_E are given by $b_E = \sum_{i=1}^{n} w_i * b_i$, $d_E = \sum_{i=1}^{n} w_i * d_i$, and $u_E = \sum_{i=1}^{n} w_i * u_i$ respectively.

5.1.2 Evaluating Knowledge

Definition 2 The *knowledge* of the truster regarding a trustee for a particular context is defined as a measure of the condition of awareness of the truster through acquaintance with, familiarity of or understanding of a science, art or technique.

The knowledge factor is made up of two parts: *direct knowledge* and *indirect knowledge.*

Direct knowledge can be formally assessed or evaluated. Indirect knowledge is more subjective. Direct knowledge can be evaluated through credentials and certificates. Indirect knowledge can be obtained by reputation. Direct knowledge and indirect knowledge are associated with triples $K_D = (b_D,\ d_D,\ u_D)$ and $K_I = (b_I,\ d_I,\ u_I)$ respectively. Each piece of direct (indirect) knowledge is categorized into positive, negative, or neutral. The elements of the triple $(b_D,\ d_D,\ u_D)$ can be computed as follows. $b_D = \frac{\text{\#positive direct knowledge + \#neutral direct knowledge/2}}{\text{total number of direct knowledge}}$. $d_D = \frac{\text{\#negative direct knowledge + \#neutral direct knowledge/2}}{\text{total number of direct knowledge}}$. If there is any direct knowledge $u_D = 0$, otherwise $u_D = 1$. Similar formulas can be written for indirect knowledge.

The weight that a truster assigns to each of these knowledge types depends on the problem context. The truster assigns the relative weights w_D, w_I for these two types of knowledge, where $w_D, w_I \in [0,1]$ and $w_D + w_I = 1$. The weights are determined by the underlying policy. Truster A's knowledge about trustee B in the context c is computed as

$$\begin{aligned} {}_AK_B^c &= w_D \times K_D + w_I \times K_I \\ &= w_D \times (b_D, d_D, u_D) + w_I \times (b_I, d_I, u_I) \\ &= (b_K,\ d_K,\ u_K) \end{aligned}$$

where $b_K = w_D \times b_D + w_I \times b_I$, $d_K = w_D \times d_D + w_I \times d_I$, $u_K = w_D \times u_D + w_I \times u_I$.

5.1.3 Evaluating Recommendation

Definition 3 A *recommendation* about a trustee is defined as a measure of the subjective or objective judgment of a recommender about the trustee to the truster.

The truster A may obtain a recommendation from multiple recommenders regarding trustee B in the context c. The goal is to generate a triple (b,d,u) from each recommender and use these to get (b_R, d_R, u_R) which represents the recommendation that A has received about B with respect to context c. First, we give the details about how the triple is computed for each recommender. Later, we describe how these results are aggregated.

Let M be one such recommender. The recommender M may or may not have a trust relationship with trustee B regarding context c. The truster A can provide a questionnaire to the recommender. The recommender is allowed to use the values +1, -1, 0, or $\perp$ in filling this questionnaire. The value +1 indicates belief, -1 indicates disbelief, 0 indicates neutral, and $\perp$ indicates unknown. The number of $\perp$s with respect to the total number of values will give a measure of uncertainty. The ratio of the number of +1s together with half the number of 0s to the total number of values gives the value for belief. The ratio of the number of -1s together with half the number of 0s to the total number of values gives the value for disbelief. If the recommender does not return a recommendation, the truster uses the triple (0,0,1) as a recommendation from M.

The truster A will have a trust relationship with the recommender M. The context of this trust relationship will be to act "reliably to provide a service (recommendation, in this case)". This trust relationship will affect the opinion of the recommendation provided by the recommender. The truster scales the recommender's opinion about the trustee with this trust value. Scaling the recommendation score based on the trust relationship between the truster and the recommender has one important benefit. Suppose that the recommender tells a lie about the trustee in the recommendation in order to gain an advantage with the truster. If the truster does not have belief on the recommender to a great degree then the belief on the recommendation will be low with the truster. Note also that if the truster disbelieves a recommender to properly provide a recommendation, it will most likely not ask for the recommendation.

The trust relationship that truster A has with trustee M in the context of providing a recommendation is represented as a 3×3 matrix. The rows of the matrix correspond to experience, knowledge, and recommendation and the columns correspond to belief, disbelief, and uncertainty. This matrix is normalized as outlined in Section 5.1.4 and converted into a triple of the form (b, d, u). This triple will be used for the scaling operation.

To do this scaling, we borrow the concept of "discounting" proposed by Jøsang [27, 28]. According to his proposition, if the recommender M disbelieves the trustee B or is uncertain about B, then A also disbelieves B or is uncertain about B to the extent scaled down by A's belief on M. Also, A's disbelief and uncertainty about M's opinion contribute towards A's uncertainty about B. If M sends the triple ${}_Mb_B,\ {}_Md_B,\ {}_Mu_B$ as a recommendation about B, and A has the trust on M as $({}_Ab_M,\ {}_Ad_M,\ {}_Au_M)$, then the *recommendation* ${}_MR_B^c$ of a recommender M for an entity B to the truster A in a context c is given by $({}_{AM}b_B^R,\ {}_{AM}d_B^R,\ {}_{AM}u_B^R)$. The values of ${}_{AM}b_B^R,\ {}_{AM}d_B^R,\ {}_{AM}u_B^R$ computed as per Jøsang's formula is:

$$\begin{aligned}
{}_{AM}b_B^R &= {}_A b_M \times {}_M b_B \\
{}_{AM}d_B^R &= {}_A b_M \times {}_M d_B \\
{}_{AM}u_B^R &= {}_A d_M + {}_A u_M + {}_A b_M \times {}_M u_B
\end{aligned}$$

Recall that the truster A may get recommendations about the trustee B from many different recommenders. Then A's belief on the recommendation about B is the average of the belief values of all recommendations and A's disbelief is the average of the disbelief values of the recommendations. The same is true for A's uncertainty about the recommendations. Therefore, if ψ is a group of n recommenders then ${}_{A\psi}b_R = \frac{\sum_{i=1}^{n} {}_{Ai}b_B^R}{n}$, ${}_{A\psi}d_R = \frac{\sum_{i=1}^{n} {}_{Ai}d_B^R}{n}$ and ${}_{A\psi}u_R = \frac{\sum_{i=1}^{n} {}_{Ai}u_B^R}{n}$. Hence, the recommendation component is expressed by the triple $({}_{A\psi}b_R,\ {}_{A\psi}d_R,\ {}_{A\psi}u_R)$.

5.1.4 Normalization of Trust Vector

Having determined the triples for each component of trust we specify the simple trust relationship between the truster A and the trustee B in a context c at time t as

$$(A \xrightarrow{c} B)_t = \begin{pmatrix} b_E & d_E & u_E \\ b_K & d_K & u_K \\ {}_{A\psi}b_R & {}_{A\psi}d_R & {}_{A\psi}u_R \end{pmatrix} \tag{5.1}$$

Given the same set of values for the factors that influence trust, two trusters may come up with two different trust for the same trustee because they may assign different weights to the different factors that influence trust. Which particular component needs to be emphasized more than the others, is a matter of trust evaluation policy of the truster. The policy is represented by the truster as a trust policy vector.

Definition 4 The *trust policy vector*, ${}_AW^c_B$, is a vector that has the same number of components as the simple-trust vector. The elements are real numbers in the range $[0,1]$ and the sum of all elements is equal to 1.

The elements of this vector are weights corresponding to the parameters of trust relationship. Let $(A \xrightarrow{c} B)_t$ be the simple trust relationship between truster A and trustee B in context c at time t. Let also ${}_AW^c_B = [W_E,\ W_K,\ W_R]$ be the corresponding trust evaluation policy vector elements such that $W_E + W_K + W_R = 1$ and $W_E, W_K, W_R \in [0,1]$. Therefore, the normalized trust relationship between a truster A and a trustee B at a time t and for a particular context c is given by

$$\begin{aligned} (A \xrightarrow{c} B)^N_t &= {}_A\mathrm{W}^c_B \times (A \xrightarrow{c} B)_t \\ &= (W_E, W_K, W_R) \times \begin{pmatrix} b_E & d_E & u_E \\ b_K & d_K & u_K \\ {}_{A\psi}b_R & {}_{A\psi}d_R & {}_{A\psi}u_R \end{pmatrix} \\ &= ({}_A\hat{b}^c_B,\ {}_A\hat{d}^c_B,\ {}_A\hat{u}^c_B) \end{aligned}$$

where ${}_A\hat{b}^c_B = W_E \times b_E + W_K \times b_K + W_R \times_{A\psi} b_R$, ${}_A\hat{d}^c_B = W_E \times d_E + W_K \times d_K + W_R \times_{A\psi} d_R$, ${}_A\hat{u}^c_B = W_E \times u_E + W_K \times u_K + W_R \times_{A\psi} u_R$.

It follows from above that each element ${}_A\hat{b}^c_B$, ${}_A\hat{d}^c_B$, ${}_A\hat{u}^c_B$ of the normalized trust relationship lies within $[0,1]$ and ${}_A\hat{b}^c_B + {}_A\hat{d}^c_B + {}_A\hat{u}^c_B = 1$.

5.2 Reasoning about Trust Relationships in Different Contexts

The model we have described so far has two shortcomings that needs to be overcome if the model is to be useful for real-world applications. First, it is not possible to compute a useful trust vector if the truster does not have any experience, knowledge, or recommendation about a trustee in a given context. The model returns the vector $(0,0,1)$ – total uncertainty. Second, the model developed so far can reason about trust relationships only with respect to a given context. In other words, it allows trust vectors to be compared only when there is an exact match on the context. These two shortcomings must be removed in order to make the trust model useful for pervasive computing applications. We remove these problems by formalizing the notion of context and describing the relationships that exist between different contexts.

Definition 5 A *context* $\mathcal{C}_i$ is represented by a set of keywords denoted by $KeywordSet_{\mathcal{C}_i}$.

Each keyword in $KeywordSet_{\mathcal{C}_i}$ is used to describe the context $\mathcal{C}_i$. The keywords in $KeywordSet_{\mathcal{C}_i}$ are semantically equivalent because they express the same context. For each context $\mathcal{C}$, we require that the $KeywordSet_{\mathcal{C}}$ should be non-empty and finite. For any two distinct contexts $\mathcal{C}$ and $\mathcal{C}'$, $KeywordSet_{\mathcal{C}} \cap KeywordSet_{\mathcal{C}'} = \{\}$. In other words, any keyword belongs to exactly one context. An example will help illustrate the notion of contexts. The context *age* can be expressed by the keywords $\{age, yearOfBirth\}$.

Consider the two contexts *doing a job* and *doing a job well*. Modeling them as distinct concepts increases the total number of contexts that must be managed. To solve this problem, we specify *doing a job* as a context and associate a set of values with it. The values in this case will be $\{badly, neutral, well\}$. Using these values, we can specify different conditions on the context. Each of these conditions represent a *derived context*. To obtain a derived context from the context $\mathcal{C}_i$, each keyword k, where $k \in KeywordSet_{\mathcal{C}_i}$, must be associated with a domain D_k that defines the set of values associated with the keyword. The formal definition of derived context appears below.

Definition 6 A *derived context* $\mathcal{DC}_i$ is one that is specified by a condition $k\ op\ v$ defined over a context $\mathcal{C}_i$ where $k \in KeywordSet_{\mathcal{C}_i}$ and $v \in D_k$ and op is a logical operator compatible with the domain of D_k.

To check whether two derived contexts specified using conditions on different keywords are equivalent, we need the notion of translation functions.

Definition 7 The *translation function* associated with a context $\mathcal{C}_i$, denoted as $TF_{\mathcal{C}_i}$, is a total function that takes as input a condition $k\ op\ v$ ($k \in KeywordSet_{\mathcal{C}_i}$) and a keyword k' ($k' \in KewordSet_{\mathcal{C}_i}$) and produces an equivalent condition defined over keyword k'. This is formally expressed as follows. $TF_{\mathcal{C}_i} : Cond_{\mathcal{C}_i} \times KeywordSet_{\mathcal{C}_i} \rightarrow Cond_{\mathcal{C}_i}$ where $Cond_{\mathcal{C}_i}$ is the set of all valid conditions specified over the keywords in $KeywordSet_{\mathcal{C}_i}$.

Since the translation function is total, for every given valid condition and keyword there exists an equivalent condition defined on the given keyword. Several steps are involved in developing the translation function. To express $k\ op\ v$ in terms of k', we need to first convert the value k to an equivalent value that is in the domain of k'. This step is performed by conversion functions which convert the value of one keyword to an equivalent value of another keyword. The second step is to convert the operator op into an equivalent operator op' that is suitable for the domain of k'. The definition of the conversion function together with the domain of the keyword can determine how the operator must be changed.

Consider the two keywords *age* and *yearOfBirth*. Suppose we want to translate $age > 18$ to an equivalent condition defined over *yearOfBirth*. The first step is to convert $age = 18$ to an equivalent value defined over *yearOfBirth*. The function that converts *age* to $yearOfBirth$ will be specified as: *yearOfBirth* = *currentYear* – *age*. For $age = 18$, this function returns $yearOfBirth = 1987$. Since $yearOfBirth$ and *age* are inversely related, (that is, *age* increases as $yearOfBirth$ decreases) the operator $>$ is inverted to obtain $<$. The results obtained by the TF_{C_i} function in this case will be $yearOfBirth < 1987$.

5.2.1 Relationships between Contexts

We now describe two kinds of relations that may exist between distinct contexts. One is the generalization/specialization relationship existing between related contexts. The other is the composition relationship between possibly unrelated contexts.

Specialization Relation

Distinct contexts may be related by the specialization relationship. The specialization relation is anti-symmetric and transitive. We use the notation $C_i \subset C_j$ to indicate that the context C_i is a generalization of context C_j. Alternately, context C_j is referred to as the specialization of context C_i. For instance, the contexts *makes decision* and *makes financial decisions* are related by the specialization relationship, that is, *makes decisions* $\subset$ *makes financial decisions*. Also, *makes financial decisions* $\subset$ *makes payment decisions*. By transitivity, *makes decisions* $\subset$ *makes payment decisions*.

Each specialization relationship is associated with a degree of specialization. This indicates the closeness of the two concepts. For instance, *makes payment decisions* is a specialization of *makes decision*, and *makes payment decisions* is also a specialization of *makes financial decisions*. However, the degree of specialization is different in the two cases. *makes payment decision* is closer to *makes financial decision* than *makes decision*. The *degree of specialization* captures this difference. Since two contexts related by specialization will not be exactly identical, the degree of specialization will be denoted as a fraction. The exact value of the fraction will be determined using domain knowledge.

Composition Relation

Specialization captures the relationship between contexts that are related. Sometimes unrelated contexts can be linked together using the composition relation. We now describe this composition relation. A context in our model can either be an *elementary* context or a *composite* context. An elementary context is one which cannot be subdivided into other contexts. A composite context is one that is composed from other contexts using the logical and operation. The individual contexts that form a composite contexts are referred to as the *component* contexts. A component context can either be composite or elementary.

We use the notation $C_i \ll C_j$ to indicate that the context C_i is a component of context C_j. In such cases, C_i is referred to as the component context and C_j is the composite context. For instance, we may have the component contexts *secure key generation* and *secure key distribution* that can be combined to form the composite context *secure key generation and distribution*. This is denoted as *secure key generation* $\ll$ *secure key generation and distribution*.

Sometimes a composite context C_i may be composed from the individual contexts C_j, C_k and C_m. All these contexts may not contribute equally to form C_i. The *degree of composition* captures this idea. A degree of composition is associated with each composition relation. Since two contexts related by composition will not be exactly identical, the degree of composition is denoted as a fraction. The sum of all these fractions equals one if C_i is composed of C_j, C_k, and C_m only. If C_i is composed of C_j, C_k, and C_m and also other component contexts, then the sum of fractions associated with C_j, C_k, and C_m must be equal to or less than one. The exact value of the fraction representing the degree of composition will be determined by domain knowledge.

Context Graphs

The specialization and the composition relations can be described using one single graph which we refer to as the *context graph*. Each node n_i in this graph corresponds to a context. There are two kinds of weighted edges in this graph: composition edges and specialization edges. A composition edge (n_i, n_j), denoted by a solid arrow from node n_i to node n_j, indicates that the context represented by node n_i is a component of the context represented by node n_j. The weight on this edge indicates what percentage of the component context comprises the composite context. A specialization edge (n_p, n_q), shown by a dashed arrow from node n_p to node n_q, indicates that the context represented by node n_p is a specialization of the context represented by node n_q. The weight on the edge indicates the degree of specialization of a context.

Unrelated contexts correspond to nodes in different context graphs. Each context corresponds to only one node in the set of context graphs. We denote the context graph associated with context C_i as $\mathcal{CG}_{C_i}$. The formal definition of a context graph is as follows.

Definition 8 A *context graph* $\mathcal{CG} = \langle \mathcal{N}, \mathcal{E}_c \cup \mathcal{E}_s \rangle$ is a weighted directed acyclic graph satisfying the following conditions.

- $\mathcal{N}$ is a set of nodes where each node n_i is associated with a context C_i and is labeled with $KeywordSet_{C_i}$. $KeywordSet_{C_i}$ is the set of keywords associated with the context C_i.

- The set of edges in the graph can be partitioned into two sets $\mathcal{E}_c$ and $\mathcal{E}_s$. For each edge $(n_i,n_j) \in \mathcal{E}_c$, the context C_i corresponding to node n_i is a component of the concept C_j corresponding to node n_j. The weight of the edge (n_i,n_j), denoted by $w(n_i,n_j)$, indicates the percentage of component context that makes up the composite context. For each edge $(n_i,n_j) \in \mathcal{E}_s$, the concept C_i corresponding to node n_i is a specialization of concept C_j corresponding to node n_j. Here again the weight of the edge (n_i,n_j), denoted by $w(n_i,n_j)$, indicates the degree of specialization.

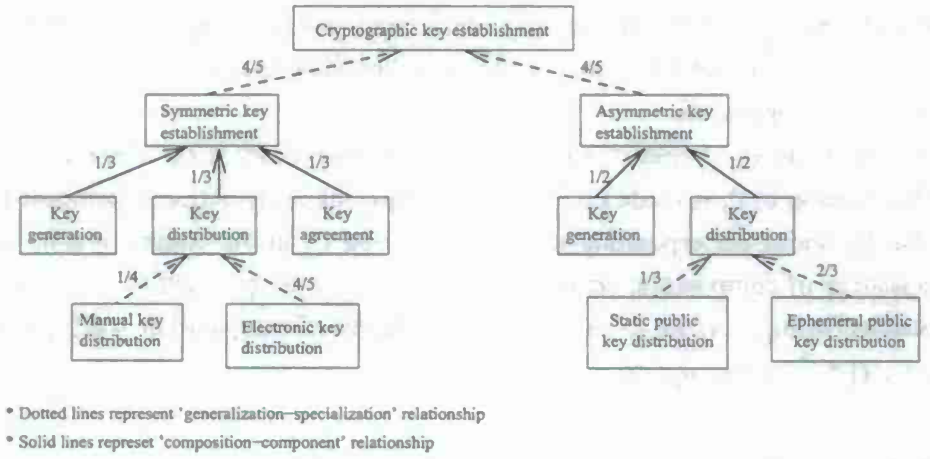

Cryptographic key establishment
4/5
4/5
Symmetric key establishment
Asymmetric key establishment
1/3
1/3
1/3
1/2
1/2
Key generation
Key distribution
Key agreement
Key generation
Key distribution
1/4
4/5
1/3
2/3
Manual key distribution
Electronic key distribution
Static public key distribution
Ephemeral public key distribution
* Dotted lines represent 'generalization–specialization' relationship
* Solid lines represet 'composition–component' relationship

Figure 5.1: Specialization and composition relationships

Figure 5.1 gives an example of a context graph that is associated with the context *cryptographic key establishment*. The solid arrows in this graph indicate composition relationships and the dashed arrows indicate generalization/specialization relationships. The context *cryptographic key establishment* can have two specializations, namely, *symmetric key establishment* and *asymmetric key establishment*. The weight on the edge connecting this *symmetric key establishment* with *cryptographic key establishment* indicates the degree of specialization. For instance, if symmetric key establishment is very closely related to key establishment, the degree of specialization may be labeled as $\frac{4}{5}$. Similarly, the edge connecting *asymmetric key establishment* to *key establishment* may be labeled as $\frac{4}{5}$. Each of these specific contexts is a composition of some component contexts. *Generation and distribution of symmetric keys* has three components – *key generation, key distribution*, and *key agreement*. A weight of $\frac{1}{3}$ can be assigned to each of these components contexts. Similarly, *generation and distribution of asymmetric keys* can have components *key generation* and *key distribution* with weights $\frac{1}{2}$ each.

A component context can also be a generalization of some specialized contexts. In the above example the context *key distribution* has two categories – *manual key distribution* and *electronic key distribution*. Similarly *key distribution* in asymmetric keys can be thought of as generalization of *static public key distribution* and *ephemeral public key distribution*.

5.2.2 Computing the Degree of Specialization and Composition

Consider two contexts c_i and c_j where $c_i \subset c_j$, that is, c_j is a specialization of c_i. The degree of specialization is computed as follows. Let n_i, n_j be the nodes corresponding to contexts c_i and c_j in the weighted graph. Let the path from n_i to n_j consisting of specialization edges be denoted as $(n_i, n_{i+1}, n_{i+2}, \ldots, n_{j-1}, n_j)$. The degree of specialization $= \Pi_{p=i}^{j-1} w(n_p, n_{p+1})$. This corresponds to our notion that the similarity decreases as the length of the path from the generalized node to the specialized node increases. Note that, in real world there may be multiple paths from c_i to c_j. In such cases, it is important that the degree of specialization yield the same values when any of these paths are used for computation.

Consider two contexts c_i and c_j such that c_j is a component of c_i. Degree of composition captures what portion of c_i is made up of c_j. The degree of composition is computed as follows. Let n_i, n_j be the nodes corresponding to contexts c_i and c_j in the context graph. Let there be m paths consisting of composition edges from n_i to n_j. Let the qth path $(1 \leq q \leq m)$ from n_i to n_j be denoted as $(n_i, n_{i_q+1}, n_{i_q+2}, \ldots, n_{j_q-1}, n_j)$. The degree of composition $= \Sigma_{q=1}^{m} (w(n_i, n_{i_q+1}) \times w(n_{j_q-1}, n_j) \times \Pi_{p=i_q+1}^{j_q-2} w(n_p, n_{p+1}))$.

5.2.3 Relationships between Context Graphs

Different information sources may use different context graphs. Comparing information or combining information that uses different context graphs may not give correct results. Before proceeding with the comparison of information obtained from different sources, the context graphs of these sources must be merged. Note that, sometimes context graphs cannot be merged because they contain conflicting information. To understand why this happens, we first need to elaborate on the relationships that can exist between a pair of context graphs. Two context graphs can be related by any of the following relationships: (i) equality, (ii) unrelated, (iii) subsumes, and (iv) incomparable.

Intuitively, two context graphs are equal if they have the same set of nodes, composition edges, and specialization edges. Moreover, each of these edges must have identical weights in the two graphs. Sometimes two context graphs are unrelated. They do not have any common context. It is conceivable that these graphs will be used for different situations. Often times two context graphs are comparable but one has more information than the other. In such cases, the context graphs are related by the subsumes relation. Often times two context graphs, neither of which subsumes the other, may be comparable. Such graphs contain different but related information. Moreover, they never have any conflicting information. Such graphs can be merged without human intervention. Two context graphs that are not unrelated are incomparable if they are not comparable. Incomparable graphs occur when the underlying assumptions are different. Since the conflicts are generated because of the differences in the underlying assumptions, they cannot be resolved without human intervention.

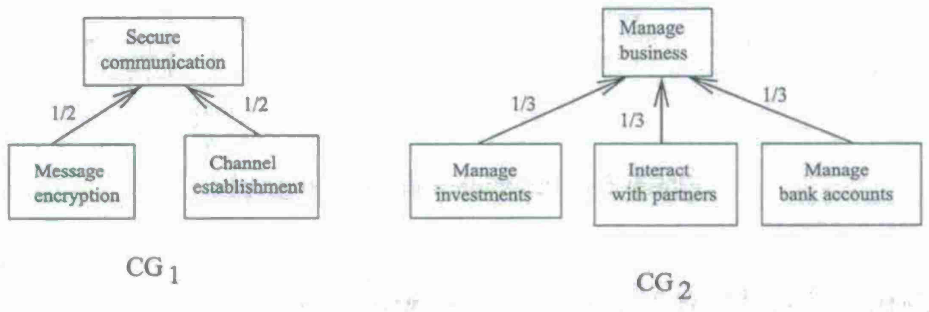


Figure 5.2: Unrelated context graphs

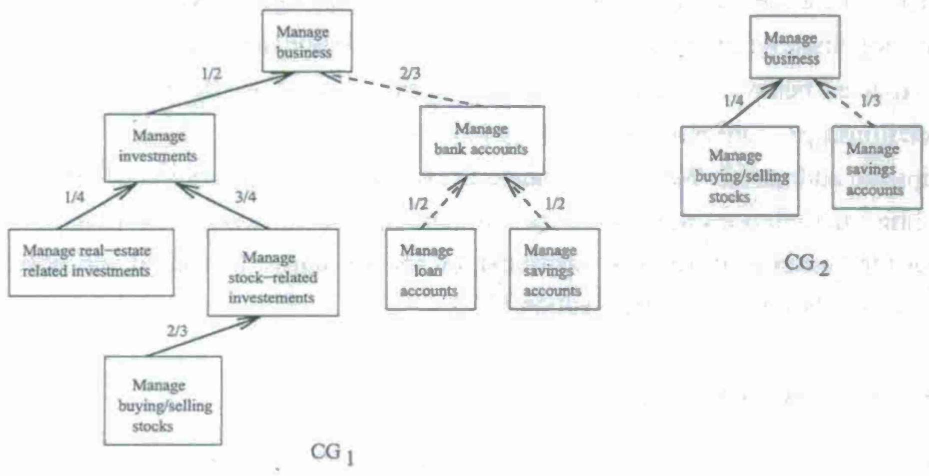
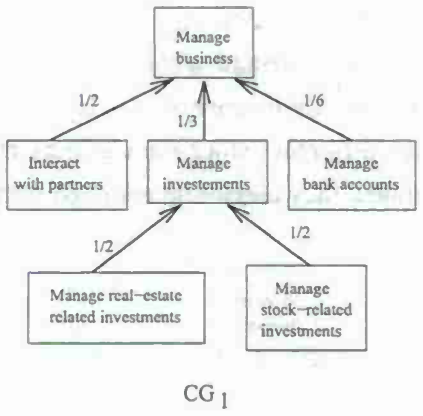


Figure 5.3: Context graphs having subsumes relation

When a truster A cannot determine the values related to his trust relationship with trustee B for a context C, the values can be obtained from one or more related contexts, say, C_i. We use the component values of the individual parameters recommendation, experience, and knowledge from C_i and use these to compute the trust vector for C. Note that, a context C may be related to many other contexts, say, C_i, C_j, and C_k. Here it is important to choose the closest related context from this set and use it for extrapolation. The details of reasoning about trust in the presence of incomplete information appears in our related papers [40, 41].



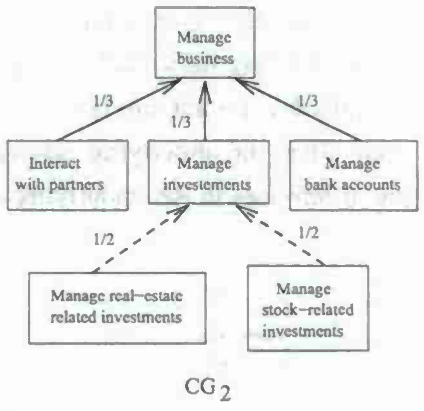


Figure 5.4: Incomparable context graphs

5.2.4 Combining Trust Vectors for Collaborations

Ad hoc collaborations such as those frequently occurring in pervasive computing applications, typically involve many cooperative entities in a relationship within a specific context. Combination of trust is needed for the interoperability of these cooperating agents. Whenever a group of agents are working together, combining their individual trust relationships is necessary to have an idea about the expected behavior of the group. Keeping this in mind we define combination operators for trust relationships. Different possibilities like one-to-many, many-to-one, and many-to-many relationships are addressed. We also formalize the effect of reconfiguration of these groups on the corresponding trust relationships. As in the comparison operation between trust relationships, we assume that the contexts of the trust relationships are the same. If needed and possible, we can extrapolate trust relationships as per section 5.2.3.

Trust relationship between a truster and a group of trustee

In real life, we often encounter situations where we have to take decisions based on information coming from different sources. Consider the scenario where an entity has existing trust relationships with different service providers for a particular service. The truster expects some service which is provided collectively by the service providers. The truster has some expectation from each individual provider. To have an idea about the service provided by the group, the combined trust of the service providers needs to be estimated. Therefore, the receiver needs a mechanism to combine the existing trust relationships to estimate an initial *composite trust relationship*. The group of service providers is considered as a single entity (trustee). Once the combination is done, the truster no longer considers the trust relationships with individual trustee. The truster begins with the combined group as a single entity and subsequently a trust relationship with the group evolves. We use the *disjunction* operator of subjective logic to define an *initial* trust relationship between a truster and the group.

Assume a truster A has trust relationships $T = (A \xrightarrow{c} B)_{t_n}^N = (b \cdot T, d \cdot T, u \cdot T)$ and $T' = (A \xrightarrow{c} C)_{t_n}^N = (b \cdot T', d \cdot T', u \cdot T')$ with two trustees B and C at the same time t_n and in the same context c. A decides to have a trust relationship with the combined group BC in the same context. as follows: $(A \xrightarrow{c} BC)_{t_n}^N = (\hat{b}, \hat{d}, \hat{u})$ where $\hat{b} = b_T + b_{T'} - b_T \times b_{T'}$, $\hat{d} = d_T \times d_{T'}$, and $\hat{u} = d_T \times u_{T'} + d_{T'} \times u_T + u_T \times u_{T'}$

Trust relationship between a group of trusters and a single trustee

Next, we address the situation where different trusters having different trust relationships with the same trustee decides to form a group. After forming the group the trusters behave as a single truster. We need to define a way to combine these different trust relationships to get the initial trust for the group. This initial trust gives the starting point of a trust relationship between the two entities. Thereafter, this trust evolves as before. But before the collaboration can succeed all trusters need to agree to a common policy as to how to continue to evaluate the trustee as a single group. In addition, the members need to agree about the following: (i) a common interval length to determine experience as well as trust, (ii) a common set of recommenders whom the group consider suitable for recommendation purposes, (iii) a common policy for evaluating trust relationships with recommenders, and (iv) a common trust evaluation policy vector to assign weights to each component. Based on this agreement each truster needs to go back and reevaluate their individual trust relationships. Let the updated trust relationships be $\hat{T} = (\hat{b}_T, \hat{d}_T, \hat{u}_T)$ and $\hat{T}' = (\hat{b}_{T'}, \hat{d}_{T'}, \hat{u}_{T'})$ respectively. We use the consensus operation in subjective logic to define the combined trust relationship between the group AB and the trustee C, as $\bar{T} = (AB \xrightarrow{c} C)_{t_n}^N = (b_{\bar{T}}, d_{\bar{T}}, u_{\bar{T}})$, where $b_{\bar{T}} = \frac{\hat{b}_T \times \hat{u}_{T'} + \hat{b}_{T'} \times \hat{u}_T}{\hat{u}_T + \hat{u}_{T'} - \hat{u}_T \times \hat{u}_{T'}}$, $d_{\bar{T}} = \frac{\hat{d}_T \times \hat{u}_{T'} + \hat{d}_{T'} \times \hat{u}_T}{\hat{u}_T + \hat{u}_{T'} - \hat{u}_T \times \hat{u}_{T'}}$, and $u_{\bar{T}} = \frac{\hat{u}_T \times \hat{u}_{T'}}{\hat{u}_T + \hat{u}_{T'} - \hat{u}_T \times \hat{u}_{T'}}$

When more than two trusters need to form a collaboration, the composite trust relationship is formed by first combining two of the trusters to form a smaller group and then enlarging the group one more truster at a time till every one of them has been included.

Trust relationship between a group of trusters and a group of trustees

We now explore the situation when a group of trusters $\mathcal{G}_r$ forms a trust relationship with a group of trustees $\mathcal{G}_e$ in some common context c. We can formalize this by combining the above two cases. Combination can take place in different ways.

1. If the group of trustees $\mathcal{G}_e$ already exists, then each truster A_i must already have, or must build a trust relationship $(A_i \xrightarrow{c} \mathcal{G}_e)_t^N$ as described in section 5.2.4. Then A_i's form the truster group $\mathcal{G}_r$ with $\mathcal{G}_e$, considering $\mathcal{G}_e$ as a single trustee, as described in section 5.2.4.
2. If the truster group $\mathcal{G}_r$ already exists with m different trust relationships like $(\mathcal{G}_r \xrightarrow{c} B_i)_t^N$ for $i = 1, 2, \ldots, m$, then $(\mathcal{G}_r \xrightarrow{c} \mathcal{G}_e)_t^N$ can be formed as in section 5.2.4.

3. If neither the group of trusters or the group of trustees exist then one of the groups has to be formed first after which then the other group is formed as explained above.

Next we examine the effect of reconfiguration of a group on the trust relationship.

Reconfiguration of a group

Ad hoc collaborations are very dynamic in nature. Consequently, anytime after a group is formed one or more members may leave or a new member may join necessitating re-evaluation of corresponding trust relationships. We have two cases to consider, namely, (i) re-evaluation owing to reorganization of trustee group, and (ii) re-evaluation owing to reorganization of truster group.

We consider the case where a new trustee, C joins an existing group of trustees G at a time t_0. For the purpose of re-evaluation of the trust relationship the truster, A, assumes the group G as a single trustee. The new trust relationship is then computed in the manner discussed in section 5.2.4. If a trustee leaves a group the re-evaluation of the trust relationship proceeds as follows. Assume C, the exiting trustee, had joined the group G at time t_0 and is leaving the group G' at time t_n. When C leaves the group it is as if a dummy trustee $\hat{C}$ with a trust relationship diametrically opposite to that of C joins the group such that the effects of C is mitigated in the group. However, at time t_n C's effective trust value has degraded from T_C to some value $T'_C = (b'_C, d'_C, u'_C)$. This is the value that needs to be mitigated. A trust relationship that is diametrically opposite to T'_C is $\hat{T}_C = (\hat{b}_C, \hat{d}_C, \hat{u}_C)$, where $\hat{b}_C = d'_C$ $\hat{d}_C = b'_C$, and $\hat{u}_C = u'_C$ The new trust relationship between the group $G' \setminus C$ is then obtained by assuming that the dummy trustee $\hat{C}$ joins the group. The exit of a group member may or may not necessitate a change in the trust evaluation policy. If the rank of the trust relationship T_C was greater than the rank of the trust relationship T_G (G being the group that C joined) then the trust evaluation policy needs to be changed after C leaves. The next re-evaluation of the trust relationship $T_{G' \setminus C}$ will be based on the new policy.

When a truster B joins an existing group of trusters G', the trust relationship is re-evaluated by considering the group G' as a single truster and then following the principles discussed in section 5.2.4. Removal of a truster from the group does not affect the group trust relationship However, the remaining group members may decide to revisit the policy for trust evaluation. The new policy will hence forth decide how the trust relationship is re-evaluated the next time.

5.3 Conclusions and Future Work

In this work, we propose a new trust model based on subjective logic for use in pervasive computing applications. We identify three parameters namely, experience, knowledge and recommendation that contribute towards defining this trust relationship. We propose expression for evaluating these factors. Next we introduce the concept of normalized trust. We show how to factor in a notion of trust policy in computing the trust vector. We also formalize the notion of trust contexts and their relationships, so that the trust model is interoperable and allows trust computation on the basis of extrapolation in the absence of enough information. Finally, we propose operators to compose trust relationships for dynamic collaboration.

A lot of work remains to be done. We plan to extend the model to support trust chains. We need to validate our model using real-world data. Finally, we plan to investigate how this model can be used to provide security services in pervasive computing application.

Chapter 6

Risk Estimation and Security Provisioning

Pervasive computing applications typically involve information flow across multiple domains. Thus, any security breach in an application can have very far-reaching consequences. Effective security mechanisms are obviously needed; however, these can be quite different from those typically deployed in conventional applications under similar circumstances. The choice of security mechanisms in pervasive environments is influenced by a number of factors. Some of the more important among these are the resource constraints of the devices, the cost of deploying security mechanisms on these devices, and the attack coverage provided by defenses. How to select the defenses is referred to as the security provisioning problem. To make matters difficult, security administrators often have to work within a fixed budget that may be less than the minimum cost of system hardening. Thus, they have to select a subset of the required security hardening measures and yet minimize the residual damage to the system caused by not plugging all required security holes. With cost-effectiveness occurring as a major factor in deciding the extent to which an organization can secure its pervasive computing environment, it is not sufficient to detect the presence or absence of a vulnerability and implement a security measure to rectify it. Rigorous analysis is required to understand the contribution of the vulnerabilities towards any possible damage to the organization's assets. Often, vulnerabilities are not exploited in isolation, but rather used in groups to compromise a system. Similarly, security policies can have a coverage for multiple vulnerabilities. Further, an attacker's perceived gains through a specific attack strategy can (and should) influence the security administrator's decision to employ a particular defense strategy. Thus, cost-effective security management requires evaluating the different scenarios that could lead to the damage of a secured asset as well evaluating the attacker's various possible attack strategies, and then come up with an optimal set of security policies (or, a defense strategy) to defend such assets.

In this chapter we formalize these issues and identify possible resolutions to some of the decision making problems related to securing a pervasive computing application. We first develop a formal model of attack trees to encode the contribution of different security conditions leading to system compromise. We formalize the notions of attack and defense strategies based on this model of attack trees. Next, we develop a model to quantify the potential damage that can occur in a system from the attack modeled by the system attack tree. We develop models of cost for defense and attack strategies. We then propose two models for the risk assessment and the security provisioning problem. The first model does not consider the attacker's strategy but treats the security administrator's problem as a multi-objective optimization problem to maximize security and minimize cost. The second model treats the security administrator's problem as a payoff problem to decide how security controls can be incorporated to maximize the return on investment under the scenario that an attacker is actively engaged in maximizing its return on attacks.

6.1 Attack Tree Model

The vulnerabilities present in a network are often exploited in groups. Materializing a threat usually requires the combination of multiple attacks exploiting different vulnerabilities. Representing different scenarios under which an asset can be damaged thus becomes important for preventive analysis. Such representations not only provide a picture of the possible ways to compromise a system, but also help to determine a minimal set of preventive actions. Given the normal operational state of a system an attack could possibly open up avenues to launch another attack, thereby taking the attacker a step closer to its goal. The presence of a vulnerability in a system does not imply that it can always be exploited. A certain state of the system in terms of access privileges, resource constraints or machine connectivities, need to be a prerequisite to be able to exploit a vulnerability. Once the vulnerability is exploited, the state of the system can change, enabling the attacker to launch the next attack in the sequence. Such a pre-thought sequence of attacks gives rise to an *attack scenario*. We capture the inter-relationships between different vulnerabilities that play together to form the basis of attacks, in the notion of attack trees. Here we briefly discuss the attack tree model. More details are available in our papers [9, 39, 10].

Different properties of the pervasive computing application effectuate different ways for an attacker to compromise a system. We first define an *attribute-template* that lets us generically categorize these system properties for further analysis.

Definition 9 An *attribute-template* is a generic property of the hardware or software configuration of a system that includes but is not limited to the following:

- *system vulnerabilities (which are often reported in the vulnerability database such as BugTraq, CERT/CC, or netcat).*
- *network configuration such as open port, unsafe firewall configuration, etc.*
- *system configuration such as data accessibility, unsafe default configuration, or read-write permission in file structures.*
- *access privilege such as user account, guest account, or root account.*

- *connectivity*
- *resource constraints*

Attribute-template lets us categorize most of the atomic properties of the system that might be of some use to an attacker. For example, *"running SSH1 v1.2.23 on FTP Server"* can be considered as an instance of the system vulnerabilities template. Similarly, *"user access on Terminal"* is an instance of the access privilege template. Such templates also lets us specify the properties in propositional logic. We define an *attribute* with such a concept in mind.

Definition 10 An *attribute* is a propositional instance of an attribute-template taking either a *true* or *false* value.

The success or failure of an attacker reaching its goal depends mostly on what truth values the attributes in the system take. Similarly, the security administrator can falsify some of the attributes using some security policies and controls to prevent an attack from succeeding. We formally define an attack tree model based on such attributes. Since we consider an attribute as an atomic property of a system, taking either a *true* or *false* value, most of the definitions are written using propositional logic involving these attributes.

Definition 11 Let S be a set of attributes. We define Att to be a mapping $Att : S \times S \to \{true, false\}$ and $Att(s_c, s_p) =$ truth value of s_p. $a = Att(s_c, s_p)$ is an *attack if* $s_c \neq s_p \wedge a \equiv s_c \leftrightarrow s_p$. s_c and s_p are then respectively called a precondition and postcondition of the attack, denoted by pre(a) and post(a) respectively.

$Att(s_c, s_p)$ is a ϕ–attack if $\exists$non-empty $S' \subset S | Att(s_c, s_p) \equiv \bigwedge_i s_i \wedge s_c \leftrightarrow s_p$ where $s_i (\neq s_c) \varepsilon S'$.

An attack relates the truth values of two different attributes to embed a cause-consequence relationship between the two. For example, for the attributes $s_c =$*"vulnerable to sshd BOF on machine A"* and $s_p =$*"root access privilege on machine A"*, $Att(s_c, s_p)$ is an attack – the sshd buffer overflow attack. We would like to clarify here that the bi-conditional logical connective "$\leftrightarrow$" between s_c and s_p does not imply that s_p can be set to *true* only by using $Att(s_c, s_p)$; rather it means that given the sshd BOF attack, the only way to make s_p *true* is by having s_c *true*. In fact, $Att($*"vulnerable to local BOF on setuid daemon on machine A"*$, s_p)$ is also a potential attack. The ϕ–attack is included to account for attributes whose truth values do not have any direct relationship. However, an indirect relationship can be established collectively. For example, the attributes $s_{c_1} =$ *"running SSH1 v1.2.25 on machine A"* and $s_{c_2} =$ *"connectivity(machine B, machine A)"* cannot individually influence the truth value of s_c, but can collectively make s_c *true*, given they are individually *true*. In such a case, $Att(s_{c_1}, s_c)$ and $Att(s_{c_2}, s_c)$ are ϕ–attacks.

Definition 12 Let A be the set of attacks, including the ϕ–attack. An *attack tree* is a tuple $AT = (s_{root}, S, \tau, \varepsilon)$, where

1. s_{root} is an attribute which the attacker wants to become *true*.
2. $S = N_{internal} \cup N_{external} \cup \{s_{root}\}$ is a multiset of attributes. $N_{external}$ denotes the multiset of attributes s_i for which $\nexists a \varepsilon A | s_i \varepsilon post(a)$. $N_{internal}$ denotes the multiset of attributes s_j for which $\exists a_1, a_2 \varepsilon A | [s_j \varepsilon pre(a_1) \wedge s_j \varepsilon post(a_2)]$.
3. $\tau \subseteq S \times S$. An ordered pair $(s_{pre}, s_{post}) \varepsilon \tau$ if $\exists a \varepsilon A | [s_{pre} \varepsilon\ pre(a) \wedge s_{post} \varepsilon post(a)]$. Further, if $s_i \varepsilon S$ and has multiplicity n, then $\exists s_1, s_2, \ldots, s_n \varepsilon S |\ (s_i, s_1), (s_i, s_2), \ldots, (s_i, s_n) \varepsilon \tau$, and
4. ε is a set of decomposition tuples of the form $\langle s_j, d_j \rangle$ defined for all $s_j \varepsilon N_{internal} \cup \{s_{root}\}$ and $d_j \varepsilon \{AND, OR\}$. d_j is *AND* when $\bigwedge_i [s_i \wedge (s_i, s_j) \varepsilon \tau] \leftrightarrow s_j$ is *true*, and *OR* when $\bigvee_i [s_i \wedge (s_i, s_j) \varepsilon \tau] \leftrightarrow s_j$ is *true*.

These set of definitions define nodes of the attack tree as propositions and edges relate the truth value of a node with that of its children. Leaf nodes on the tree represent propositions related to the different system states, which may be *true* or *false* depending on what defenses are in place. The truth values of the leaf nodes progressively define if the propositions on the internal nodes would be *true* or *false*. If no defense is installed, all leaf nodes would be *true*. This would finally lead to the root node to become *true* as well. In such a case, the attacker is assumed to have successfully met its objective. Due to the presence of the *AND-OR* decompositions, the root node may become *true* even if all leaf nodes are not *true*. Similarly, all leaf nodes need not be *false* for the root to become *false*.

Fig. 6.1 shows an example attack tree, with the attribute "*root access on machine A*" as s_{root}. The multiset S forms the nodes of the tree. The multiset $N_{external}$ resemble the leaf nodes of the tree. These nodes reflect the initial vulnerabilities present in a network and prone to exploits. Since, an attribute can be a precondition for more than one attack, it might have to be duplicated, hence forming a multiset. The attribute "*machine A can connect to machine B*" in the example is one such attribute. The set of ordered pairs, τ, reflect the edges in the tree. The existence of an edge between two nodes imply that there is a direct or indirect relationship between their truth values, signified by the decomposition at each node. The *AND* decomposition at a node requires all child nodes to have a truth value of *true* for it to be *true*. The *OR* decomposition at a node requires only one child node to have a truth value of *true* for it to be *true*. Using these decompositions, the truth value of an attribute $s_j \varepsilon N_{internal} \cup \{s_{root}\}$ can be evaluated after assigning a set of truth values to the attributes $s_i \varepsilon N_{external}$.

6.2 Defense and Attack Strategies

In order to defend against possible attacks, the system administrator can choose to implement a variety of safeguard technologies. Each choice of action can have a different cost involved. Some measures may have multiple coverages, but with higher costs. The defender has to make a decision

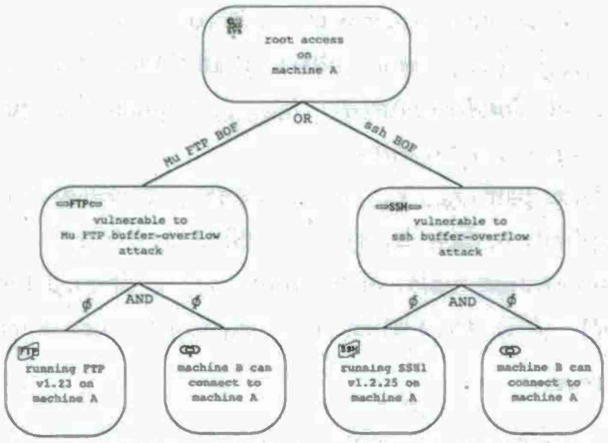

root access on machine A
OR
Mu FTP BOF
ssh BOF
==FTP==
vulnerable to Mu FTP buffer-overflow attack
==SSH==
vulnerable to ssh buffer-overflow attack
AND
ϕ
ϕ
AND
ϕ
ϕ
FTP
running FTP v1.23 on machine A
machine B can connect to machine A
SSH
running SSH1 v1.2.25 on machine A
machine B can connect to machine A

Figure 6.1: Example attack tree

and choose to implement a subset of these policies in order to maximize the resource utilization.

Definition 13 Given an attack tree $(s_{root}, S, \tau, \varepsilon)$, the mapping $D : N_{external} \rightarrow \{true, false\}$ is a *defense* if $\exists s_i \in N_{external} |\ D(s_i) = false$.

We use the term *security control* synonymously to indicate a defense. A defense is nothing but a preventive measure to falsify one or more leaf nodes thereby stopping an attacker from reaching its goal. Further, in the presence of multiple defenses D_k, the truth value of an attribute $s_i \in N_{external}$ is taken as $\bigwedge_k D_k(s_i)$. Given a defense D, the set of all $s_i \in N_{external} |\ D(s_i) = false$ is called the *coverage* of D. Hence, for a given set of defenses, we can define the *coverage matrix* specifying the coverage of each defense.

Definition 14 For a given set of d defenses, the *defense strategy* $\vec{S_D} = (S_{D_1}, S_{D_2}, \ldots, S_{D_d})$ is a boolean vector indicating which defenses are chosen by the defender. $S_{D_i} = 1$ if defense D_i is chosen, zero otherwise.

The choice of this vector specifies which leaf nodes in the attack tree would be *false* to begin with. An attacker typically exploits leaf nodes that are not covered by any defense in order to progressively climb up the tree, inflicting some amount of damage to the network at every step. However, it is not always correct to assume that an attacker can no longer exploit some parts of the attack tree because of the installed defenses. With the appropriate tools and knowledge, an attacker may have the potential to bypass a defense as well. In other words, leaf nodes that were made *false* by a defense can be reverted back to being *true*. We thus assume an attacker with the proper knowledge to able to breach a defense. However, in order to do so, the attacker will have to incur some cost, often related to the number of defenses in place and the difficulty to breach them. If an attacker's gains are less than the cost incurred, then its effort to breach the defense is not worth the time and value. This primarily motivates the defender to still install defenses despite there being a chance of breach.

Given that the attacker can bypass an installed defense (after incurring a cost), it can start its exploits from any leaf node on the attack tree. The attacker's progress towards the root is then decided by the leaf nodes it chooses. Note that choosing all leaf nodes that can collectively make an intermediate node *true* need not always be the best approach for the attacker. For instance, given that defenses will be in place at different levels of the tree and the attacker will have to incur a cost to bypass them, it is possible that the attacker derives more payoff by inflicting damages at different parts of the attack tree rather than continuing along a single scenario all the way up to the root. An attack strategy is thus defined as follows.

Definition 15 Let a denote the number of unique leaf nodes in an attack tree. An *attack strategy* $\vec{S_A} = (S_{A_1}, S_{A_2}, \ldots, S_{A_a})$ is a boolean vector indicating which leaf nodes in the tree are chosen by the attacker for exploit. $S_{A_i} = 1$ if node $A_i \in N_{external}$ is chosen, zero otherwise.

An attack strategy specifies the path(s) that the attacker pursues to an intermediate level or the top level of the attack tree. The success of the strategy depends on the defense strategy adopted by the defender, as well as the number of levels it can move up on the tree. Another way to visualize an attack strategy is the set of leaf nodes that the attacker assumes to be *true*, or will make *true* by breaching the defenses protecting them.

6.3 Cost Model

In order to defend against attacks, a security manager can choose to implement a variety of safeguard technologies. Each of these comes with different costs and coverages. For example, to defend against the ftp/.rhost exploit, one may choose to apply a security patch, disable the FTP service, or simply tighten the write protection on the .rhost directory. Each choice of action can have a different cost. Besides, some measures have multiple coverages, but with higher costs. A security manager has to make a decision and choose to implement a subset of these policies in order to maximize the resource utilization. However, this decision is not a trivial task. Security planning begins with risk assessment which determines threats, loss expectancy, potential safeguards and installation costs.

The potential damage, P_j, represents a unitless damage value that an organization might have to incur in the event that an attribute s_j becomes *true*. Based on Butler's multi-attribute risk-assessment framework [6, 7], we specify below the four steps to calculate the potential damage for an attribute s_j.

Step1: Identify potential consequences of having a *true* value for the attribute, induced by some attack. In our case, we have identified five outcomes – lost revenue ($$$), non-productive downtime (hrs), damage recovery ($$$), public embarrassment (severity scale) and law penalty (severity scale) – denoted by $x_{1j}, x_{2j}, x_{3j}, x_{4j}$ and x_{5j}.

Step2: Estimate the expected number of attack occurrence, $Freq_j$, resulting in the consequences. A security manager can estimate the expected number of attack from the organization-based historical data or public historical data.[1]

Step3: Assess a single value function, $V_{ij}(x_{ij})$, for each possible consequence. The purpose of this function is to normalize different unit measures so that the values can be summed together under a single standard scale.

$$V_{ij}(x_{ij}) = \frac{x_{ij}}{\underset{j}{Max}\, x_{ij}} \times 100 \qquad , 1 \le i \le 5 \tag{6.1}$$

Step4: Assign a preference weight factor, W_i, to each possible consequence. The weight factor represents an organization's concerns for different outcomes. A security manager can rank each outcome on a scale of 1 to 100. The outcome with the most concern would receive 100 points. The manager ranks the other attributes relative to the first. Finally, the ranks are normalized and set as W_i.

The potential damage for the attribute can then be calculated from the following equation.

$$P_j = Freq_j \times \sum_{i=1}^{5} W_i V_{ij}(x_{ij}) \tag{6.2}$$

When using an attack tree, a better quantitative representation of the cost is obtained by considering the residual damage once a set of security policies are implemented. Hence, we augment each attribute in the attack tree with a value signifying the amount of potential damage residing in the subtree rooted at the attribute and the attribute itself.

Definition 16 Let $AT = (s_{root}, S, \tau, \varepsilon)$ be an attack tree. An *augmented-attack tree* $AT_{aug} = AT|\langle I, V\rangle$ is obtained by associating a tuple $\langle I_i, V_i\rangle$ to each $s_i \varepsilon S$, where

1. I_i is an indicator variable for the attribute s_i, where

$$I_i = \begin{cases} 0 & , if\ s_i\ is\ false \\ 1 & , if\ s_i\ is\ true \end{cases}$$

2. V_i is a value associated with the attribute s_i.

In this work, all attributes $s_i \varepsilon N_{external}$ are given a zero value. The value associated with $s_j \varepsilon N_{internal} \cup$

[1]Also known as an incident report published annually in many sites such as CERT/CC or SANS.ORG.

$\{s_{root}\}$ is then computed recursively as follows.

$$V_j = \begin{cases} \sum_{k|(s_k,s_j)\varepsilon\tau} V_k + I_j P_j & , if\ d_j\ is\ AND \\ \underset{k|(s_k,s_j)\varepsilon\tau}{Max}\ V_k + I_j P_j & , if\ d_j\ is\ OR \end{cases} \tag{6.3}$$

Ideally, P_j is same for all identical attributes in the multiset. We took a *"panic approach"* in calculating the value at each node, meaning that given multiple subtrees are rooted at an attribute with an *OR* decomposition, we choose the maximum value. The residual damage of the augmented tree is then defined as follows.

Definition 17 Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and a vector $\vec{T} = (T_i)$, $T_i \varepsilon \{0,1\}; 1 \leq i \leq m$, the *residual damage* is defined as the value associated with s_{root}, i.e.,$RD(\vec{T}) = V_{root}$

Similar to the potential damage, the security manager first lists possible security costs for the implementation of a security control, assigns the weight factor on them, and computes the normalized value. The only difference is that there is no expected number of occurrence needed in the evaluation of security cost. In our case, we have identified five different costs to implementing a security control – installation cost ($$$), operation cost ($$$), system down-time (hrs), incompatibility cost (scale), and training cost ($$$). The overall cost C_j, for the security control SC_j, is then computed in a similar manner as for potential damage, with an expected frequency of 1. The total security cost for a set of security controls implemented is then defined as follows.

Definition 18 Given a set of *m* defenses, each having a cost $C_i; 1 \leq i \leq m$, and a vector $\vec{T} = (T_i)$, $T_i \varepsilon \{0,1\}; 1 \leq i \leq m$, the *defense strategy cost* is defined as $SCC(\vec{T}) = \sum_{i=1}^{m} (T_i C_i)$

From these two models of defense strategy cost and residual damage we can formulate the system administrator's decision problem as finding a defense strategy that minimizes the defense strategy cost as well as minimize the residual damage. To perform this optimization, the system administrator needs to express a preference – whether to give more emphasis on the defense strategy cost or on the residual damage. Depending on this preference the decision problem will provide the best defense strategies at specific cost levels. By setting the residual cost to zero, the system administrator can come up with a defense strategy that guarantees complete system safety under closed world assumption (that is no zero day attack). However, owing to budget constraints, the system administrator may need to understand better the tradeoff between defense strategy cost and network safety. Additionally, the system administrator may also want to determine optimal, yet robust defense strategies. These are strategies that can tolerate some (predetermined) degree of failures (that is, compromise from attacks) yet continue to ensure no further residual damage. Such strategies provide some latitude against zero day attacks.Towards this end, we formulate the first security provisioning problem as a multi-objective robust optimization problem.

The Security Provisioning Problem as a Multi-objective Robust Optimization Problem

Let $\vec{T} = (T_i)$ be a boolean vector. A perturbed assignment of radius r, $\vec{T}_r$, is obtained by inverting the value of at most r elements of the vector $\vec{T}$. The value r determines how many security controls may fail before the system is compromised. The robust optimization problem can then be defined as follows: Given an augmented-attack tree $(s_{root}, S, \tau, \varepsilon)|\langle I, V\rangle$ and m security controls, find a defense strategy $\vec{T^*} = (T_i^*)$, $T_i^* \varepsilon \{0,1\}; 1 \le i \le m$, which minimizes the total security control cost and the residual damage, satisfying the constraint $\max_{\vec{T_r^*}} RD(\vec{T_r^*}) - RD(\vec{T^*}) \le D$ where, D is the maximum perturbation allowed in the residual damage.

6.4 Payoff Model

To factor in the attacker's strategy, we observe that the cost of realizing an attack strategy is related to the effort that the attacker has put forward in overcoming any defenses put forward. We model this cost under an assumption that stronger defenses are likely to have a higher cost of implementation. Under this assumption, we measure the relative difficulty to breach a defense – a value in $[0,1]$ – and assign the cost to breach it, $BC(\cdot)$, as a fraction (given by the difficulty value) of the cost of implementation of the defense, i.e.$BC(D_i) = \frac{C_i}{\underset{i}{Max}\, C_i} \times C_i$

Definition 19 Given a set of d defenses, a defense strategy $\vec{S_D}$ and an attack strategy $\vec{S_A}$ on an attack tree AT, the *attack strategy cost ASC* is defined as $ASC(\vec{S_D}, \vec{S_A}) = \sum_{i=1}^{d} \sum_{j|D_i(A_j)=false} [BC(D_i) S_{D_i} S_{A_j}]$

The expression above iterates through the leaf nodes covered by a particular defense. Thereafter, the cost to breach the defense is added to the attack strategy cost if the defense is part of the defense strategy and the leaf node is part of the attack strategy. When a breach occurs, the cost paid by the defender to install it (C_i) is a loss, called the *breach loss* $BL(\cdot)$ and expressed in a manner similar to the above equation. $BL(\vec{S_D}) = \sum_{i=1}^{d} \sum_{j|D_i(A_j)=false} [C_i S_{D_i} S_{A_j}]$

We then define the defender and attacker payoffs as follows.

Definition 20 For a given defense strategy $\vec{S_D}$ and an attack strategy $\vec{S_A}$ on an augmented-attack tree AT_{aug}, the *defender's payoff POD* is given as, $POD(\vec{S_D}, \vec{S_A}) = DI(\vec{0}, \vec{1}) + DSC(\vec{S_D}) - DI(\vec{S_D}, \vec{S_A}) - BL(\vec{S_D})$ and the *attacker's payoff POA* is given as, $POA(\vec{S_D}, \vec{S_A}) = DI(\vec{S_D}, \vec{S_A}) - ASC(\vec{S_D}, \vec{S_A})$

Here, $DI(\vec{0}, \vec{1})$ signifies the maximum damage possible on the attack tree. This happens when there are no defenses installed and the attacker exploits all leaf nodes. $\vec{0}$ represent the all zero vector and $\vec{1}$ is the all one vector. Note that both payoff functions employ the same DI value derived from the attack tree. More details can be found in our paper [10].

To account for differences arising in the magnitudes of the values of the POD and ASC functions, we normalize them. The normalized functions for POD and POA – in the range of $[0,1]$ –

are then given as,

$$POD_{norm}(\vec{S_D},\vec{S_A}) = \frac{POD(\vec{S_D},\vec{S_A})}{DSC(\vec{S_D}) + DI(\vec{0},\vec{1})}$$

$$POA_{norm}(\vec{S_D},\vec{S_A}) = \frac{POA(\vec{S_D},\vec{S_A}) + ASC(\vec{S_D},\vec{S_A})}{ASC(\vec{S_D},\vec{S_A}) + DI(\vec{0},\vec{1})}$$

The normalized versions are more intuitive in understanding the payoff functions model. The defender has an investment worth $DSC(\vec{S_D}) + DI(\vec{0},\vec{1})$ on the attack tree. POD_{norm} gives the fraction of this investment protected by the defender's strategy for a particular attack strategy. In other words, POD_{norm} gives the fractional return on investment for the defender. From an attacker's perspective, the best it can do is gather the payoff from maximum damage and also retain the cost incurred while doing so to itself. $DI(\vec{S_D},\vec{S_A})$ is the amount that it actually derives. POA_{norm} is thus the fractional return on attack to the attacker.

The defender's optimization problem is to find a defense strategy $\vec{S_D}$ that gives maximum POD_{norm} under all possible attack strategies. The attacker's optimization problem is to find an attack strategy $\vec{S_A}$ that gives maximum POA_{norm} under all possible defense strategies. We want to emphasize here that solving just one problem is not sufficient. For example, assume that the defender has found the optimal solution to its problem. The POD_{norm} reported by the solution implicitly assumes that the attacker will launch the strategy $\vec{S_A}^*$ that gives the highest attacker payoff – established in the optimization problem by the relation. If the attacker also solves its own optimization problem, there is no guarantee that the best strategy found by it is the same $\vec{S_A}^*$ as found in solving the defender's optimization problem. The outcome in this case could be that both the attacker and the defender get sub-optimal payoffs. This implies the requirement to solve both problems simultaneously, the desired solution being the so called Nash equilibrium in game theory parlance [32]. The equilibrium defense and attack strategy pair $\vec{S_D}^*$ and $\vec{S_A}^*$ satisfy the conditions $POD_{norm}(\vec{S_D}^*,\vec{S_A}^*) > POD_{norm}(\vec{S_D},\vec{S_A}^*)$ and $POA_{norm}(\vec{S_D}^*,\vec{S_A}^*) > POA_{norm}(\vec{S_D}^*,\vec{S_A})$ for any given defense strategy $\vec{S_D}(\neq \vec{S_D}^*)$ and attack strategy $\vec{S_A}(\neq \vec{S_A}^*)$.

6.5 Conclusion

We addressed the security provisioning problem in pervasive computing environment, namely, how to select a subset of security hardening measures from a given set so that the total cost of implementing these measures is not only minimized but also within budget and, at the same time, the cost of residual damage is also minimized. We developed two models to address this problem – a initial simplistic model that does not consider the attacker's perceptions about cost to break the system, and a second one that includes this cost.

In a related work [11], which we do not include here for lack of space, we show how workflow profiles can be used to capture the contexts in which a communication channel can be used in a pervasive environment. We formulate a set of constrained multi-objective optimization problems that minimize the residual damage and the maintenance cost incurred to keep the workflow secure and running.

Both these models take a static approach to security provisioning. There is however a dynamic aspect to the security planning process. For every attack, there is a certain probability of occurrence that can change during the life time of a system depending on what the contributing factors for the attack are and how they are changing. During run time, the system administrator may need to revise her decision based on such emerging security conditions. The next step in this work is to model this dynamic aspect. We have developed a preliminary model to address this problem that appears in our paper [38].

Chapter 7

Controlled Disclosure of Location Information

Pervasive computing applications typically use mobile devices that capture the locations of the user. The location information is used to provide better services. Often such applications need continuous location-based services (LBS) where the mobile object must periodically communicate its location to the service provider. One serious concern is the potential usage of of the location data to infer sensitive personal information about the mobile users. With access to the location data, sender anonymity can be violated even without the capability to track a mobile object. We refer to this class of adversaries as *location-unaware adversaries*. Such adversaries use external information to perform attacks resulting in restricted space identification, observation identification and location tracking [17].

Location obfuscation is one of the widely researched approaches to safeguard location anonymity. This technique guarantees that the location data received at the LBS provider can be associated back to more than one object – to at least k objects under the location *k-anonymity* model [17]. For this, a cloaking region is communicated to the service provider instead of the actual location. A k-anonymous cloaking region contains at least $k-1$ other mobile objects besides the service user. However, this approach is not sufficient to preserve privacy in a continuous LBS. In the continuous case, an object maintains an ongoing session with the LBS, and successive cloaking regions may be correlated to associate the session back to the object. Such session associations reveal the trajectory of the involved object, and any sensitive information thereof. Assuring that every cloaking region contains k objects is not sufficient since the absence of an object in one of the regions eliminates the possibility that it is the session owner. Performing such elimination is much easier for a *location-aware adversary* who has the capability to monitor users. This class of adversaries has exact location information on one or more objects and uses it to eliminate possibilities and probabilistically associate the session to consistently existing objects.

Session association attacks can be avoided if it can be assured that every cloaking region in a

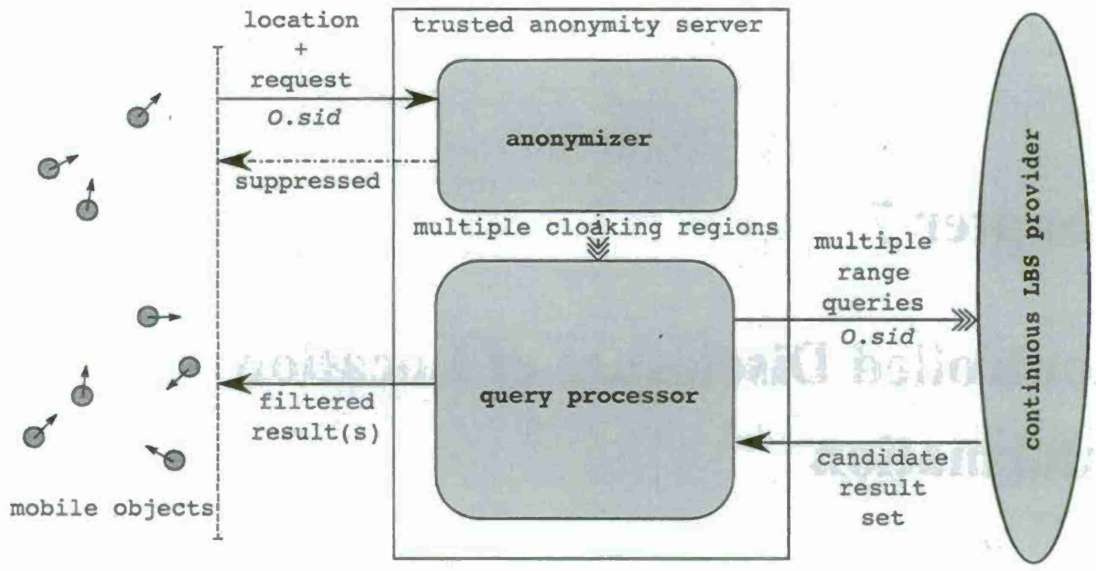


Figure 7.1: Schematic of the system architecture

session contains k common objects. This is referred to as *historical k-anonymity* [5]. However, as a result of the movement of objects, a historically k-anonymous cloaking region is very likely to grow in size over time, thereby deteriorating service quality. In this work, we make an attempt to identify the issues involved with effectively enforcing historical k-anonymity.

The rest of the chapter is organized as follows. Section 7.1 presents the architecture of our system and some observations on historical k-anonymity. Section 7.2 describes our anonymization algorithm *CANON* that enforces historical k-anonymity. Section 7.3 compares the performance of *CANON* with that of an existing one, namely, *ProvidentHider*, that also provides historical k-anonymity. Section 7.4 concludes the paper.

7.1 System Architecture

Figure 7.1 depicts our system consisting of three layers – (i) mobile objects, (ii) a trusted anonymity server, and (iii) a continuous LBS provider. The trusted anonymity server acts as a channel for any communication between mobile objects and continuous LBS providers. A mobile object O initiates a service session by registering itself with the anonymity server. The registration process includes the exchange of current location information ($O.loc$) and service parameters signifying the request to forward to the LBS provider, as well as the anonymity level ($O.k$) to enforce while doing so. The anonymity server issues a pseudo-identifier and uses it both as a session identifier ($O.sid$) with the mobile object and as an object identifier when communicating with the LBS provider. A set of cloaking regions is then generated for the requesting object and multiple range queries are issued to the LBS provider for these regions. Communication between the anonymity

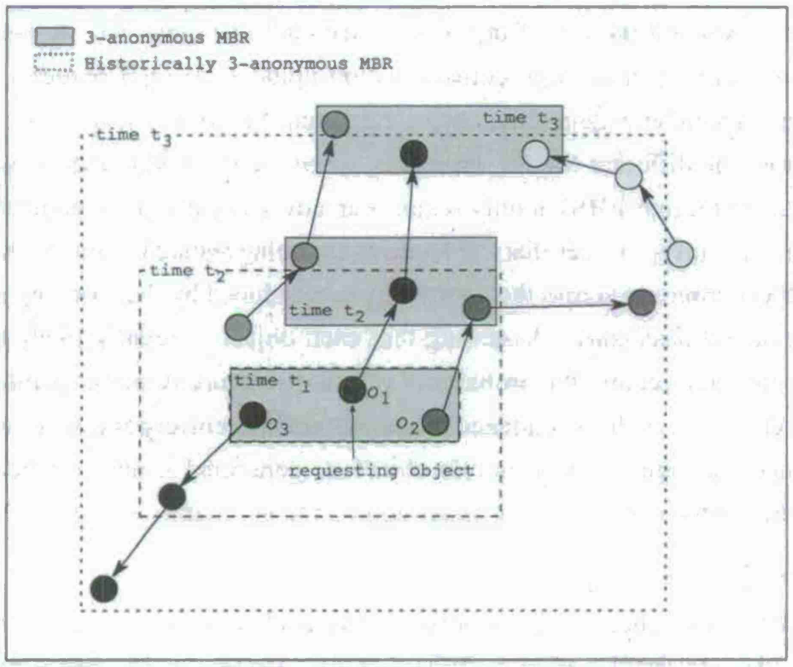


Figure 7.2: Conventional k-anonymity and historical k-anonymity.

server and the LBS provider is always referenced using the object identifier so that the LBS can maintain service continuity. The candidate results retrieved from the LBS provider are filtered at the anonymity server and then communicated to the mobile object. Subsequent location updates from the mobile object are handled in a similar fashion (with the pre-assigned session identifier) until the anonymity level cannot be satisfied or the service session is terminated. A request is suppressed (dropped) when the anonymity requirements can no longer be met within the same service session. A new identifier is then used if the mobile object re-issues the same request. We further assume that an object does not change its service parameters during a session. A separate session is started if a request with different service parameters is to be made. Therefore, an object can have multiple sessions running at the same time, each with a different session identifier.

7.1.1 Historical k-anonymity

The primary purpose of a cloaking region is to make a given mobile object O indistinguishable from a set of other objects. This set of objects, including O, forms the *anonymity set* of O. Objects in the anonymity set shall be referred to as *peers* of O and denoted by O*.peers*. A cloaking region for O is usually characterized by the minimum bounding rectangle (MBR) of the objects in O*.peers*. Larger anonymity sets provide higher privacy, while at the same time can result in reduced service quality owing to a larger MBR. Therefore, the cloaking region is typically required to achieve an acceptable balance between anonymity and service quality.

Consider the movement pattern of the objects depicted in Figure 7.2. A 3-anonymous MBR is computed for O_1 during three consecutive location updates. If O_1's requests at the three time instances are mutually independent from each other, then the privacy level of O_1 is maintained at 3-anonymity across the different MBRs. However, when the same identifier is associated with all the MBRs (as in a continuous LBS), it only requires an adversary the knowledge of O_1, O_2 and O_3's positions at time t_1, t_2 and t_3 to infer that the requests are being issued by object O_1. This is because O_1 is the only object common across the anonymity sets induced by the cloaking regions. We refer to this as a case of *full disclosure*. Assuming that each object is equally likely to be included in another object's cloaking region, the probability of full disclosure is unacceptably high.

Remark 1: Let $A_1, \ldots, A_n$ be a sequence of anonymity sets corresponding to $n > 1$ consecutive k-anonymous cloaking regions for a mobile object O, generated from a collection of N mobile objects. Then, the probability that the intersection of the anonymity sets $\mathcal{S}_n = \bigcap_i A_i$ has at least p objects, $p > 1$, is $\left(\prod_{i=1}^{p-1} \frac{k-i}{N-i}\right)^n$.

Remark 2: If $k \leq \frac{N+1}{2}$ then the probability of full disclosure is at least $\frac{3}{4}$. The full disclosure risk is given as $\mathcal{D}_{full} = Pr(|\mathcal{S}_n| = 1) = Pr(|\mathcal{S}_n| \geq 1) - Pr(|\mathcal{S}_n| \geq 2)$. Since intersection of the anonymity sets contain at least one object, we have $Pr(|\mathcal{S}_n| \geq 1) = 1$. Hence, $\mathcal{D}_{full} = 1 - (\frac{k-1}{N-1})^n$. With $k \leq \frac{N+1}{2}$, or $\frac{k-1}{N-1} \leq \frac{1}{2}$, we have $D_{full} \geq 1 - \frac{1}{2^n} \geq 1 - \frac{1}{2^2} = \frac{3}{4}$.

We also observe in Figure 7.2 that it does not require knowledge on the objects' locations at all three time instances in order to breach O_1's privacy. In fact, location knowledge at time instances t_1 and t_2 is sufficient to lower O_1's privacy to 2-anonymity. This is referred to as a *partial disclosure*. Such disclosures occur when the intersection of anonymity sets (corresponding to the same object) contain less than the desired number of peers (the anonymity level k).

A straightforward extension of the conventional k-anonymity model that can counter risks of full and partial disclosures in a continuous LBS is to ensure that all anonymity sets within a service session contain at least k common objects.

Remark 3: **Historical k-anonymity.** Let $A_1, \ldots, A_n$ be a sequence of anonymity sets corresponding to the cloaking regions with the same identifier and at time instants $t_1, \ldots, t_n$, $t_i > t_j$ for $i > j$, respectively. The anonymity set A_i is then said to satisfy historical k-anonymity if $|A_1 \cap \ldots \cap A_i| \geq k$.

In other words, the sequence of anonymity sets preserve historical k-anonymity if all subsequent sets after A_1 contain at least k same objects from A_1. Figure 7.2 depicts how the cloaking regions should change over time in order to ensure that object O_1 always has historical 3-anonymity.

7.1.2 Implications

Historical k-anonymity impedes session association attacks by location-aware adversaries. However, maintaining acceptable levels of service can become increasingly difficult in case of historical

k-anonymity. We have identified three issues for consideration that impact the practical usage of historical k-anonymity.

Defunct peers: A defunct peer in an anonymity set is an object that is no longer registered with the anonymity server. As a result, it can no longer be ascertained that a cloaking region includes the peer. If the first cloaking region generated during a particular session contains exactly k objects, then every other anonymity set in that session must contain the same k objects for it to be historically k-anonymous. A defunct peer in this case does not allow subsequent cloaking regions to satisfy historical k-anonymity and introduces possibilities of partial disclosure.

Diverging peer trajectories: The trajectories of peers influence the size of a cloaking region (satisfying historical k-anonymity) over time. Refer to Figure 7.2. The MBR for object O_1 becomes increasingly larger owing to the trajectory of object O_3. Bigger cloaking regions have a negative impact on service quality. In general, the more divergent the trajectories are, the worse is the effect. Algorithms that use a maximum spatial resolution will not be able to facilitate service continuity as spatial constraints will not be met.

Locality of requests: The significance of a particular service request can often be correlated with the locality where it originated. For instance, let us assume that the region shown in Figure 7.2 corresponds to an urban locality. Further, object O_1 issues a request to periodically update itself with information (availability, price, etc.) on the nearest parking garage. At time instance t_1, an adversary cannot infer which object (out of O_1, O_2 and O_3) is the actual issuer of the request. However, as O_3 moves away from the urban locality (suspiciously ignoring the high concentration of garages if it were the issuer), an adversary can infer that the issuer of the request is more likely to be O_1 or O_2. We say that these two objects are still in the locality of the request. If historical k-anonymity is continued to be enforced, O_3 (and most likely O_2 as well) will be positioned in different localities, thereby allowing an adversary infer with high confidence that O_1 is the issuer of the request. Note that these three issues are primarily applicable in the context of a continuous LBS.

7.2 The CANON Algorithm

We propose CANON which is an anonymization algorithm that enforces historical k-anonymity for use with a continuous LBS. An overview of this algorithm is given by Procedure 1.

CANON is initiated by the anonymity server whenever it receives a request from a mobile object O. The algorithm starts by first checking if O has an open session with respect to the current request. If it finds one then the set of peers is updated by removing all defunct peers from the set. Otherwise, a peer set is generated for O through a procedure *CreatePeerSet* and a session identifier is assigned. The newly generated (or updated) peer set must have at least $O.k$ objects in order to continue to the next step; otherwise the request is suppressed and the session is terminated.

Procedure 1 CANON(Object o)

Input: Mobile object o (includes all associated data).

Output: A set of peer groups (one of them includes o); **null** if request is suppressed (cannot satisfy anonymity).

```
1: if ($o.sid$ = null) then
2:     $o.peers = CreatePeerSet(o)$
3:     $o.sid$ = new session identifier
4: else
5:     remove defunct objects in $o.peers$
6: end if
7: if ($|o.peers| < o.k$) then
8:     $o.sid$ = null
9:     return null
10: end if
11: $peerGroups = PartitionPeerSet(o)$
12: if ($\exists g \in peerGroups$ such that $|g| < 2$) then
13:     $o.sid$ = null
14:     return null
15: end if
16: return $peerGroups$
```

If the number of peers, $o.peers$ generated by *CreatePeerSet* is less than $o.k$, then the algorithm terminates as historical k-anonymity cannot be ensured. The next step is to divide the peer set into groups over which the range queries will be issued. A *peer group* is defined as a subset of $o.peers$. *PartitionPeerSet* divides $o.peers$ into disjoint peer groups. Each peer group defines a smaller cloaking region than that defined by the entire peer set and reduces the impact of diverging trajectories on service quality. The peer groups returned by CANON are used to issue multiple range queries (one for each) with the same object identifier. Finally, the algorithm checks that each group contains at least two objects in order to avoid the disclosure of exact location information to location-unaware adversaries.

7.2.1 Handling defunct peers

As mentioned earlier, defunct peers can influence the lifetime of a service session by reducing the peer set size to below the limit that satisfies historical k-anonymity. The resolution is to include more than k objects in the first peer set. This is achieved in CANON as follows. It uses an *oversize factor* τ that relatively specifies the number of extra peers that must be included in the peer set. The minimum initial size of the peer set of an object o is equal to $(1+\tau) \times o.k$ with this strategy. We say "minimum" because other parameters introduced later can allow more peers to be included. Note that since CANON partitions the peer set into further groups before issuing a query, the area of the cloaking region defined by the enlarged peer set has little or no influence on service quality.

However, we would still not want the area to expand extensively in order to curb the issue of request locality.

7.2.2 Deciding a peer set

The *CreatePeerSet* procedure determines the initial peer set for an object. At this point, we need to ensure that majority of the objects in the peer set are in the locality of the request. We believe there are two requirements to address in this regard.

1. Objects in the peer set should define an area where the request is *equally significant* to all the peers.
2. Objects in the peer set should move so that the defined area does not expand *too much*.

The first requirement will prohibit the inclusion of peers that are positioned in a locality where the issued request is unlikely to be made. The second requirement addresses locality of requests in the dynamic scenario where the trajectories of the peers could be such that they are positioned in very different localities over time. Preventing the MBR of the peer set from expanding prohibits peers from being too far away from each other. The first requirement can be fulfilled by choosing peers according to the Hilbert Cloak algorithm. Peers chosen according to Hilbert indices will induce a small MBR, thereby ensuring that they are more likely to be in the same locality. However, a peer set generated by this process cannot guarantee that the second requirement will be fulfilled for long. This is because the neighbors of an object (according to Hilbert index) may be moving in very different directions.

It is clear from the above observation that the direction of travel of the objects should be accounted for when selecting peers. The direction of travel is calculated as a vector from the last known location of the object to its current location, i.e. if $O.loc_1 = (x_1, y_1)$ and $O.loc_2 = (x_2, y_2)$ are the previously and currently known positions of O respectively, then the direction of travel is given as $O.dir = O.loc_2 - O.loc_1 = (x_2 - x_1, y_2 - y_1)$. $O.dir$ is set to $(0,1)$ (north) for newly registered objects. A θ-*neighborhood* for O is then defined as the set of all objects whose direction of travel is within an angular distance θ (say in degrees) from $O.dir$. Therefore, a 0°-neighborhood means objects traveling in the same direction, while a 180°-neighborhood contains all objects. If all peers are chosen within a 0°-neighborhood then it is possible that the area defined by the initial peer set will more or less remain constant over time. However, the initial area itself could be very large due to the non-availability of such peers within a close distance. On the other hand, using a 180°-neighborhood essentially allows all objects to be considered and hence the area can be kept small by including close objects. Of course, the area may increase unwantedly over time. Peer set generation is therefore guided by two system parameters in CANON - the *neighborhood step size* θ and the *full-MBR resolution* α_{full}. The neighborhood step size specifies the resolution at which the θ-neighborhood is incremented to include dissimilar (in terms of travel direction) peers.

Procedure 2 CreatePeerSet(Object O)

Input: Mobile object O (includes all associated data), and system globals τ, θ and α_{full}.
Output: A set of peer objects (including O).

```
1: $\mathcal{L}$ = set of available mobile objects sorted by their Hilbert index
2: $k_{of} = (1+\tau) \times O.k$; $\mathcal{P} = \phi$
3: repeat
4:    $\mathcal{L}_c = \phi$
5:    for all ($l \in \mathcal{L}$ in order) do
6:       if ($|\mathcal{L}_c| \geq k_{of}$ and $AreaMBR(\mathcal{L}_c \cup \{l\}) > \alpha_{full}$) then
7:          break
8:       end if
9:       $\mathcal{L}_c = \mathcal{L}_c \cup \{l\}$
10:   end for
11:   $\mathcal{P}_{prev} = \mathcal{P}$; $f = 1$; $O_{pivot}$ = first object in $\mathcal{L}_c$
12:   repeat
13:      $\mathcal{P} = (f\theta)$-neighbors of $O_{pivot}$ in $\mathcal{L}_c$
14:      $f = f + 1$
15:   until ($|\mathcal{P}| \geq \min(k_{of}, |\mathcal{L}_c|)$)
16:   $\mathcal{L} = \mathcal{L} - \mathcal{P}$
17: until ($O \in \mathcal{P}$)
18: if ($|\mathcal{P}| < k_{of}$) then
19:    $\mathcal{P} = \mathcal{P} \cup \mathcal{P}_{prev}$
20: else if ($|\mathcal{L}| < k_{of}$) then
21:    $\mathcal{P} = \mathcal{P} \cup \mathcal{L}$
22: end if
23: return $\mathcal{P}$
```

The full-MBR resolution specifies some area within which the issued request is equally likely to have originated from any of the included objects, thereby making it difficult for an adversary to eliminate peers based on position and request significance. For small values of θ and some α_{full}, all objects in a peer set would ideally move in a group, in and out of a locality. Procedure 2 outlines the pseudo-code of *CreatePeerSet*. We assume the existence of a function *AreaMBR* that returns the area of the minimum bounding rectangle of a set of objects.

CreatePeerSet first creates a sorted list $\mathcal{L}$ of all registered objects according to their Hilbert indices. It then continues to divide them into buckets (starting from the first one in the sorted list) until the one with O is found. Every time a bucket is formed, $\mathcal{L}$ is updated by removing all objects in the bucket from the list. We now describe how to get a set $\mathcal{L}_c$ of candidate objects that can potentially form a bucket. Starting from the first available object in $\mathcal{L}$, we continue to include objects in $\mathcal{L}_c$ as long as the minimum peer set size (denoted by k_{of} and decided by the oversize factor) is not met, or the area of the MBR of included objects is within the full-MBR resolution. Note that, as a result of this condition, the minimum required size of the peer set receives more prominence than the resulting area. Hence, the full-MBR resolution is only a guiding parameter and not a constraint. Next, the algorithm selects k_{of} objects from the candidate set to form a

Procedure 3 PartitionPeerSet(Object $\mathcal{O}$)

```
Input: Mobile object $\mathcal{O}$ (includes all associated data) and system global $\alpha_{sub}$.
Output: A set of peer groups.
1: Sort objects in $\mathcal{O}.peers$ by their Hilbert index
2: $peerGroups = \phi$
3: $bucket = \phi$
4: for all ($l \in \mathcal{O}.peers$ in order) do
5:    if ($AreaMBR(bucket \cup \{l\}) \leq \alpha_{sub}$) then
6:       $bucket = bucket \cup \{l\}$
7:    else
8:       $peerGroups = peerGroups \cup \{bucket\}$
9:       $bucket = \{l\}$
10:   end if
11: end for
12: $peerGroups = peerGroups \cup \{bucket\}$
13: return $peerGroups$
```

bucket. The first object in $\mathcal{L}_c$ is chosen as a pivot and all objects in the θ-neighborhood of the pivot are included in the bucket. If the bucket is not full up to its capacity (k_{of}) and more objects are present in $\mathcal{L}_c$, then the neighborhood is increased by the step size θ. By the end of this process, the bucket would either contain k_{of} objects or there are less than k_{of} objects in $\mathcal{L}_c$. The latter is only possible when list $\mathcal{L}$ contains less than k_{of} objects, i.e. the last bucket is being created. Once the bucket with $\mathcal{O}$ is found, two more checks are required. First, if $\mathcal{O}$'s bucket has less than k_{of} objects (possible if it is the last one), then it is merged with the previous bucket. Second, if the number of objects remaining in $\mathcal{L}$ is less than k_{of} (implying $\mathcal{O}$'s bucket is second to last), then the remaining objects are included into $\mathcal{O}$'s bucket.

CreatePeerSet uses θ-neighborhoods and the full-MBR resolution to balance between dissimilar peers and the resulting MBR area. While the step size θ allows incremental selection of dissimilar peers, α_{full} guides the extent of increment admissible to generate a localized peer set. Note that the creation of a peer set is a one time procedure every service session. Hence, a good estimation of the direction of travel is required to avoid diverging trajectories. CANON uses an instantaneous direction vector, since we believe this method performs reasonably well in road networks.

7.2.3 Handling a large MBR

The full-MBR resolution parameter is used to control breaches related to request localities. Typical values are in the range of 10 to 50 km^2. The parameter is therefore not intended to help generate cloaking regions with small MBRs. A continuous LBS would require a much finer resolution to deliver any reasonable service. Further, depending on variations in velocity and the underlying road network, some extent of expansion/contraction of the MBR is very likely. The MBR of a peer set is therefore not a good candidate to issue the range queries. Instead, the peer set is partitioned into multiple disjoint groups by *PartitionPeerSet*. Partitioning of the peer set eliminates empty spaces between peers (introduced in the first place if trajectories diverge) and produces smaller MBRs for the range queries [49]. This partitioning is done such that each peer group has a maximum spatial resolution. In CANON, the maximum spatial resolution of a peer group is specified as the *sub-MBR resolution* α_{sub}. α_{sub} is relatively much smaller than α_{full}. Procedure 3 outlines the partitioning method.

The partitioning is performed in a manner similar to Hilbert Cloak, with the difference that each bucket now induces an area of at most α_{sub} instead of a fixed number of objects. Starting from the first object in the Hilbert-sorted peer set, an object is added to a bucket as long as the sub-MBR resolution is met; otherwise the current bucket is a new peer group and the next bucket is created. We do not handle the case when a peer group contains only one object. Our CANON algorithm checks that such groups do not exist (safeguard against location-unaware adversaries); otherwise the request is suppressed. However, the partitioning algorithm itself can relax the sub-MBR resolution when a peer group with a single object is found. The manner in which such relaxations can be done will be addressed in a future work.

7.3 Empirical Study

The experimental evaluation compares the performance of CANON with the *ProvidentHider* algorithm. For every new request, *ProvidentHider* first groups all available objects from a Hilbert-sorted list such that each bucket holds $O.k$ objects; more if adding them does not violate a maximum perimeter (P_{max}) constraint. The peer set of an object is the bucket that contains the object. A range query is issued over the area covered by the objects in the peer set only if the maximum perimeter constraint is satisfied; otherwise the request is suppressed. Refer to [31] for full details on the algorithm. We measure a number of statistics to evaluate the performance.

- *service continuity:* average number of requests served in a session
- *service failures:* percentage of suppressed requests
- *safeguard against location-unaware adversaries:* average size of the peer group to which the issuing object belongs

We have generated trace data using a simulator [14] that operates multiple mobile objects based on real-world road network information available from the National Mapping Division of the US Geological Survey. We have used an area of approximately $168\,km^2$ in the Chamblee region of Georgia, USA for this study.

The used traffic volume information (Table 7.1) results in 8,55 8 objects with 34% on expressways, 8% on arterial roads and 58% on collector roads. The trace data consists of multiple records

Table 7.1: Mean speed, standard deviation and traffic volume on the three road types.

road type	traffic volume	mean speed	standard deviation
expressway	2916.6 cars/hr	90 km/hr	20 km/hr
arterial	916.6 cars/hr	60 km/hr	15 km/hr
collector	250 cars/hr	50 km/hr	10 km/hr

spanning one hour of simulated time. A record is made up of a time stamp, object number, x and y co-ordinates of object's location, and a status indicator. The status indicator signifies if the object is registered to the anonymity server. An object's status starts off randomly as being active or inactive. The object remains in the status for a time period drawn from a normal distribution with mean 10 minutes and standard deviation 5 minutes. The status is randomly reset at the end of the period and a new time period is assigned. The granularity of the data is maintained such that the Euclidean distance between successive locations of the same object is approximately 100 meters. Each object has an associated k value drawn from the range $[2,50]$ by using a Zipf distribution favoring higher values and with the exponent 0.6. The trace data is sorted by the time stamp of records.

During evaluation, the first minute of records is used only for initialization. Subsequently, the status of each record is used to determine if the object issues a request. Only an active object is considered for anonymization. If the object was previously inactive or its prior request was suppressed, then it is assumed that a new request has been issued. Otherwise, the object is continuing a service session. The anonymizer is then called to determine the cloaking region(s), if possible. The process continues until the object enters an inactive (defunct) state. Over 2,000,000 anonymization requests are generated during a pass of the entire trace data.

Default values of other algorithm parameters are set as follows: $\tau = 0.0$, $\alpha_{full} = 25\,km^2$, $\alpha_{sub} = 1\,km^2$, $\theta = 180°$ and $P_{max} = 5000\,m$. A $5000\,m$ perimeter constraint for *ProvidentHider* is approximately an area of $1.6\,km^2$. Compared to that, α_{sub} has a smaller default value. The precision is around $1000\,m$ (assuming a square area) which serves reasonably well for a Pay-As-You-Drive insurance service. The full-MBR resolution of $25\,km^2$ evaluates to a locality about $\frac{1}{32}^{th}$ the size of New York City. The entire map is assumed to be on a grid of $2^{14} \times 2^{14}$ cells (a cell at every meter) while calculating the Hilbert indices [30]. Objects in the same cell have the same Hilbert index.

The following points summarize the results from the experimental study. The detailed analysis appears in our related paper [12].

- CANON has a superior performance compared to *ProvidentHider* in maintaining longer service sessions across a wide range of anonymity requirements. More requests are also successfully anonymized by CANON.

- Including a small number of extra objects in a peer set is advantageous in handling defunct peers. However, extremely large peer sets can be detrimental.

- Use of direction information during the formation of a peer set does help avoid peers drifting away from each other over time. Choice of a too small neighborhood affects service quality, but is not necessary to balance performance across different measures.

- Performance is better with larger sub-MBR resolutions. However, performance in high precision services may be improved with a good strategy to relax the constraint.

- Service continuity is marginally different for different full-MBR resolutions. However, failure to serve new requests is much lower with smaller resolutions.

7.4 Conclusion and Future Work

Owing to the limitations of *k*-anonymity in a continuous LBS, an extended notion called historical *k*-anonymity has been recently proposed for privacy preservation in such services. However, all known methods of enforcing historical *k*-anonymity significantly affects the quality of service. In this paper, we identified the factors that contribute towards deteriorated service quality and suggested resolutions. We proposed the CANON algorithm that delivers reasonably good service quality across different anonymity requirements. The algorithm uses tunable parameters to adjust the size of a peer set, trajectories of peers and cloaking regions over which range queries are issued. Immediate future work includes optimizing the performance of CANON in terms of better usage of directional information. We believe this optimization is crucial in order to have similar performance across all levels of anonymity requirements. Merging location anonymity and query privacy in a continuous LBS is a natural extension of this work.

Chapter 8

Designing Secure Pervasive Computing Applications

Pervasive computing applications are extremely complex; they have to satisfy functional as well as non-functional requirements, such as security. Moreover, security requirements are not confined to one module of the application, but must be consistently applied to all the modules. Due to the complexity of such applications, security cannot be added as an afterthought but must be addressed during the very early stages while the application is being designed.

In this work, we provide a methodology for designing such applications and getting assurance that the security properties are indeed satisfied. Often times, a security property can be satisfied by multiple security solutions. The solutions may differ with respect to the amount of protection they offer, the cost, the resource constraints, and other parameters. In such cases, we demonstrate how to do a trade-off analysis to identify the security solution that best meets the project goals.

The rest of the chapter is organized as follows. Section 8.1 presents an overview of our aspect-oriented risk-driven methodology. Section 8.2 discusses in details our security analysis and trade-off analysis techniques. Section 8.3 illustrates our approach using an example e-commerce application. Section 8.4 concludes our paper with a pointer towards future directions.

8.1 Aspect-Oriented Risk-Driven Development Methodology

Pervasive computing applications are exceedingly complex. Complex software is not designed as a monolithic unit, but it is decomposed into modules on the basis of functionality. Security concerns are not confined to one module of the application but impact its multiple components. Thus, security solutions used for thwarting these attacks must be consistently applied across these various components. We advocate the use of aspect-oriented methodologies for designing secure pervasive computing application. Aspect-oriented methodologies provide a modular approach to developing and reasoning of such cross-cutting features that impact multiple components.

We use the Unified Modeling Language (UML) to specify our models as it is the de facto software specification language used in the industry. Specifically, we represent our models using UML 2.0 [34]. The models typically consist of both static class diagrams and dynamic behavior diagrams. We demonstrate dynamic behavior specified as sequence diagrams, but these diagrams are not a requirement of the techniques we use. We find that sequence diagrams are especially convenient when dealing with behavior such as security protocols, and since our examples use such protocols we have chosen to utilize sequences in our modeling and analyses.

Our aspect-oriented methodology is composed of several steps. The process starts with system architects and designers creating models that describe the functionality of the system. We refer to these functional models as *primary models*. Note that, the primary models describe just the functionality of the system – the security mechanisms have not yet been incorporated into these models.

Once we have an initial functional design, designers must perform a risk assessment of the application. During this process, the system stakeholders (e.g. end users, designers, developers, and management) identify sensitive system assets which can be targeted by attackers. Different stakeholders can place different values on an asset, so the stakeholder and the value they assign to a particular asset are both needed in our methodology. Designers must develop security requirements for these assets and identify threats against them, with the aid of security standards such as ISO 14508: Common Criteria [19] and ISO/IEC 13335-5: Guidelines for Management of IT Security [21]. Designers and security experts must also rank the threats. Designers also identify potential security solutions that can mitigate specific risks, as part of the assessment process.

We model the attacks as aspects as they are not confined to one module of the application. Similarly, security solutions are also modeled as aspects. Aspects make it easier for designers to understand, manage and change these models separately. Since models are developed separately, a library of reusable attack and security solution models is feasible. Our work uses two types of aspects. A *generic aspect* is reusable across applications and it can be thought of as a template that must be instantiated. It is specified using parameterized notations. We instantiate a generic aspect by binding its parameters to elements in the primary model to create a *context-specific aspect*.

We compose context-specific aspects with primary models to create design models in which the aspect has been integrated. In order for composition to produce a meaningful model, the models being composed must be specified at similar levels of abstraction. However, we do not require any particular level of abstraction in our techniques and tools. Therefore, designers can compose and analyze a set of models at different levels of abstraction to produce different kinds of information, depending on the amount of detail available at a particular point in the design cycle.

The composition of primary model with the different aspects yield different types of models. We compose an attack model with a primary model to create what we term a *misuse model*. The analysis of a misuse model reveals the extent to which the primary model may be compromised through application of a successful attack. Composing a security mechanism with a primary model yields a *security-treated model*. A security-treated model represents a system in which some security solution has been incorporated into the primary model. In a similar fashion, composing an attack model with a security-treated model yields a *security-treated misuse model*. Analysis of security-treated misuse model reveals the efficacy of the security solution in protecting against the given attack.

Often times, multiple security solutions may protect against a given attack. In such cases, we need to evaluate which solution best meets the project and security goals. It may be difficult for a designer to determine how different parts of the system, designed to meet different goals, interact with each other. Performing security analysis in the context of the whole system can help a designer understand these interactions better. Performing trade-off analysis can help a designer make informed choices when faced with multiple designs that mitigate security threats equally well. AORDD trade-off analysis allows designers to analyze various security design solutions against properties such as required security levels, and project constraints such as time-to-market, budget, and resource constraints *at the same time*, in a single trade-off analysis.

8.2 AORDD Analysis

We approach analysis in AORDD in two steps. First, we perform a formal security analysis to give assurance that the system, created by integrating a security solution model, is indeed resilient to the targeted attack. We transform a UML misuse model into Alloy and use the Alloy Analyzer [25] to reason about its security properties. The results of the analysis either give assurance that the security properties exist, or show that they do not. The second step in AORDD analysis is to compute a BBN trade-off analysis network. BBN is a powerful technique for reasoning under uncertainty, using disparate information [16]. Input to the BBN consists of the evidence from the security analysis, risk information from other AORDD steps, and trade-off parameters. The trade-off analysis computes a fitness score, showing how well the proposed security solution meets the project goals. However, project-specific goals are rarely static over the course of system development, so our BBN topology allows designers to easily change parameters and priorities in real time as they explore candidate security solutions.

Figure 8.1 shows an UML activity diagram that describes the steps in an iteration of AORDD analysis. The solid circle and outlined solid circle represent the initial and final states (respectively) of an AORDD analysis. Ovals are activities (four in this diagram), and rectangles are objects produced or consumed during the activity. Solid arrows show control flow while dashed arrows show flow of objects among activities. The dashed arrow into the Security Solution Treatment Level parameters object indicates that information from Analyzer results is needed by it. The first

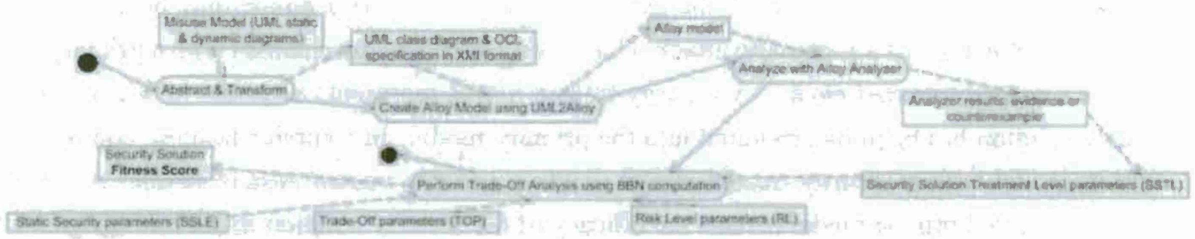

Misuse Model (UML static & dynamic diagrams)
UML class diagram & OCL specification in XMI format
Alloy model
Analyze with Alloy Analyser
Abstract & Transform
Create Alloy Model using UML2Alloy
Analyzer results: evidence or counterexample
Security Solution Fitness Score
Perform Trade-Off Analysis using BBN computation
Security Solution Treatment Level parameters (SSTL)
Static Security parameters (SSLE)
Trade-Off parameters (TOP)
Risk Level parameters (RL)

Figure 8.1: Steps in AORDD security and trade-off analysis

three activities produce an evaluation of the security provided by a security solution to protect against a successful attack. The fourth activity results in a fitness score for the security solution, with respect to security and other trade-off parameters.

Based on the results of the security evaluation, a designer may decide to iterate the security analysis steps with a different security solution prior to performing any trade-off analysis. Similarly, based on the fitness score, a designer might decide to iterate the trade-off analysis, changing the priority of trade-off parameters, or relaxing some of them. In practice, security acceptance criteria are often relaxed in the face of budget and/or time-to-market constraints. Relaxing constraints can have a great effect on fitness score.

8.2.1 Security Analysis

We use the UML2Alloy tool to transform a UML model into Alloy. Its input consists of a UML class diagram in XML Metadata Interchange (XMI) format [35] , and an accompanying OCL [37] specification of behavior. We therefore begin with the Abstract & Transform activity as the first activity in AORDD analysis. This activity takes as input a UML misuse model that a user creates by composing an attack model with either a system model or a security-treated system model. A designer must abstract the misuse model to only include elements associated with testing the security properties of interest. We use a UML CASE tool, ArgoUML [3], to create the UML class diagram and OCL specification. ArgoUML, like most UML tools, allows us to export the model in XMI format.

The next activity, Create Alloy Model using UML2Alloy, applies UML2Alloy to the XMI representation. UML2Alloy implements transformation rules to create an Alloy model [1, 2]. This model is input to the next activity, Analyze with Alloy Analyzer. The Alloy Analyzer searches the state space exhaustively on all possible valid instances of the model, up to the user-specified scope, for a counterexample. The output from the analyzer must be interpreted by a human, and be input into the BBN topology via computer assistance. If a counterexample is produced, the input to the BBN should reflect that the security solution does not provide adequate protection. Otherwise, the input represents the analysis assurance that the security solution included in the misuse model provides protection against the attack.

8.2.2 Trade-Off Analysis

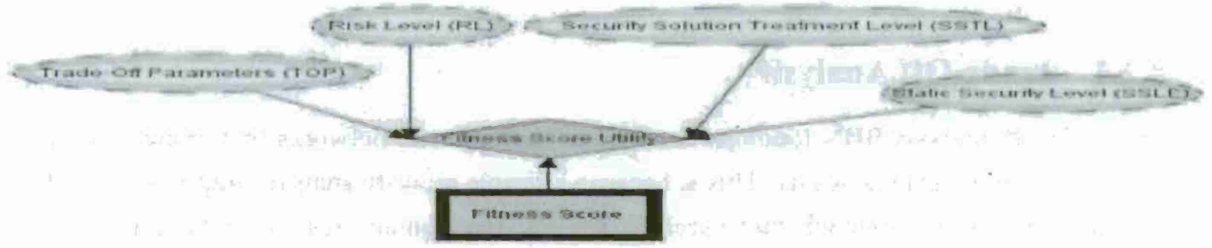
Our trade-off analysis BBN topology consists of multiple sub-networks that relate to a security solution and to security analysis. This is because a simple security analysis output is not sufficient for a designer to determine whether a security solution is adequate. Analysis either proves a particular successful attack path (misuse) to be executable, or provides evidence that it is not executable. However, the existence of an attack path does not imply that the attack will actually happen. It means that there exists a possibility of an attack. A successful attack depends on other factors, such as the likelihood or frequency of the attack, and the mean time and effort needed to launch a successful attack. These latter characteristics in turn depend on the skills, motivation and resources of the attacker [20]. Our trade-off analysis takes these characteristics into consideration, along with the impact of a successful attack on the value of system assets. We also include the project-specific consequence of incorporating a security solution to prevent the attack, such as development cost and time, in the form of variables.

Our trade-off topology consists of four sub-networks. The subnets are shown as object inputs to Perform Trade-Off Analysis using BBN Computation activity in Figure 8.1. The information categories represented by the trade-off subnets are the static security level variables (SSLE), risk level variables (RL), the security solution treatment level variables (SSTL), and the trade-off parameters (TOP).

The SSLE variables represent information regarding the criticality of the system assets, along with stakeholder asset value information, that system designers obtain from the risk assessment process. The RL variables represent information regarding identified security risks. Designers obtain part of this information during risk assessment, and part through security analysis of an initial system misuse model. Recall that risk assessment is a required step of the AORDD methodology that must be performed prior to any analysis.

SSTL variables represent information relevant for measuring the abilities of a security solution to prevent the attack, along with development and maintenance costs. Again, designers obtain part of this information through security analysis, and part from the risk assessment process. The TOP variables consist of relevant project goals and their relative priorities. This information comes from various project stakeholders and decision makers. The trade-off parameters are used to compute a fitness score that reflects the ability of the security solution to meet the set of trade-off goals.

We use the Hugin tool [18] to specify and compute the trade-off topology. We present BBN diagrams and computations in this paper using output from this tool. Figure 8.2 shows a Hugin representation of the top-level portion of the topology. Each oval section of the topology is a sub-network. The topology computes a decision variable Fitness Score (rectangle with thick border) for a security solution using four subnets (ovals with dotted and thick outlines). Subnet values and


Risk Level (RL)
Security Solution Treatment Level (SSTL)
Trade Off Parameters (TOP)
Static Security Level (SSLE)
Fitness Score Utility
Fitness Score
Figure 8.2: Trade-off analysis using BBN

a decision variable utility (diamond) are used to compute the score.

8.3 Example E-Commerce Application

We applied our proposed approach on an example e-commerce platform called ACTIVE. ACTIVE provides services for electronic purchasing of goods over the Internet [13]. The IST EU-project CORAS [47] performed three risk assessments of ACTIVE in the period 2000-2003. The risk assessment performed by the IST EU-project CORAS demonstrated that the ACTIVE login service is vulnerable to man-in-the-middle attack, which allows an attacker to intercept information that may be confidential. The man-in-the-middle attack may be passive or active. During a passive attack, the attacker eavesdrops on the message flow between a requestor and authenticator. By contrast, an attacker participates in the communication during an active attack: changing, deleting, or inserting messages between the requestor and authenticator.

8.3.1 Identifying Threats to the E-Commerce Application

In order to understand the impact a man-in-the-middle attack has on the e-commerce login service, we need to generate a misuse model. The misuse model is obtained by composing the primary model with the man-in-the-middle attack. Figure 8.3 shows portions of a primary model and a generic aspect, in the form of sequence diagrams. The generic attack model in Figures 8.3(b) and (c) specify a passive attack; messages pass through the attacker, but are not changed prior to forwarding.

The portion of a primary model in part (a) of Figure 8.3 shows two classes, *ActiveClient* and *LoginManager*. A message is sent from *ActiveClient* to execute the *requestLoginPage* method in *LoginManager*. The result of this operation returns a *loginPage* message to *ActiveClient*. *Active-Client* then executes an internal method, *ProcessPage*. A portion of a generic man-in-the-middle attack aspect model is shown in parts (b) and (c) of Figure 8.3. There are three classes, *|Sender*, *|Attacker*, and *|Receiver*. The | symbol at the beginning of any name in the generic aspect model

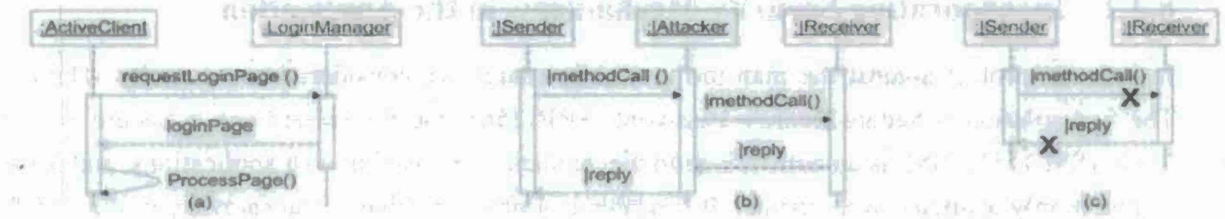


Figure 8.3: (a): Primary model, (b) and (c): generic man-in-the-middle aspect

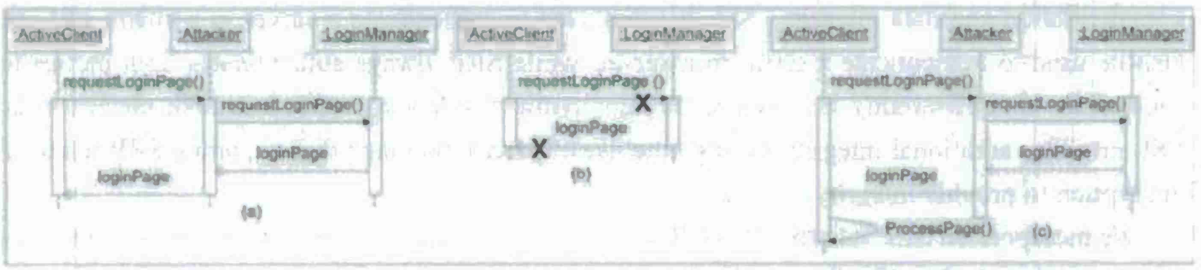


Figure 8.4: (a), (b): Context-specific man-in-the-middle, (c): composed model

serves as an indicator that this element is a parameter that can be bound to elements in the primary model that are of the same UML type, prior to model composition. The generic aspect (b) shows a message to execute a method called *|methodCall* to be sent to *|Attacker*, and from *|Attacker* to *|Receiver*. There is a response, *|reply* that is sent back. Part (c) shows behavior that is not allowed (indicated by an *X* mark), that is, some message or reply going directly between *|Sender* and *|Receiver*. Our composition techniques allow us to specify such elements that will be removed prior to composition if they exist.

We specify bindings of generic aspect parameters to primary model elements of the same type, then instantiate the aspect to create a context-specific aspect model, which we compose with the primary model. For example, using the models in Figure 8.3, we can specify that *|Sender* should be bound to *ActiveClient*, *|Receiver* should be bound to *LoginManager*, *|methodCall* should be bound to *requestLoginPage*, and *|reply* to *loginPage*. There is no corresponding primary model element to *|Attacker*, so our tools automatically create a binding from *|Attacker* to *Attacker*. The context-specific attack model is shown in Figures 8.4(a) and (b).

The context-specific attack model is then composed with primary model using our model composition techniques [] to generate the misuse model. Portion of the misuse model appears in Figure 8.4(c). The analysis of the misuse model indicates that the man-in-the-middle attack is indeed possible in the ACTIVE system.

8.3.2 Incorporating Security Mechanisms in the Application

In order to protect against the man-in-the-middle attack, we considered two security solutions. The first solution is Secure Remote Password (SRP) [56], and the second one is Secure Sockets Layer (SSL) [55]. SSL is an authentication mechanism often used in web applications, and is part of commonly available web clients. It operates just above a reliable transport layer (e.g. TCP). SRP is an alternative mechanism that is not generally available at lower levels of communication, and must be added at the application level. Both mechanisms provide user authentication, data confidentiality, and data integrity. SSL is often used to authenticate a server to a client, and can also be used to authenticate a client to a server, while SRP always authenticates both parties to each other. Confidentiality is provided through symmetric key encryption in both mechanisms. SSL provides additional integrity through the use of hashed message digests, while SRP relies on encryption to provide integrity.

We incorporated SRP into the ACTIVE e-commerce system. The security treated system must now be analyzed. Specifically, we checked whether the security treated system is prone to man-in-the-middle attacks. We generated the misuse model for the SRP treated system by composing the context-specific man-in-the-middle attack with the SRP treated e-commerce system. We then analyzed the misuse model to check for the possibility of attacks. The analysis involved several steps. First, we pruned the security treated model to remove the parts that were not pertinent to the analysis. Note that, all the models we have discussed so far including the abstracted model are represented in UML and OCL. To automate the analysis, we converted the abstract security treated misuse model into Alloy using UML2Alloy tool. The security properties that needed verification in the security treated misuse model were formalized in OCL, which were converted into Alloy assertions by the UML2Alloy tool. The Alloy Analyzer did not find any property violation, so no counterexample was produced for a scope of 20. This proved that incorporating SRP into ACTIVE helped protect it against man-in-the-middle attack. We applied the same approach on SSL. However, in this case, our analysis produced a counterexample, showing that the SSL security treated model is not effective in protecting against active man-in-the middle attacks. However, it is resilient against passive man-in-the-middle attacks.

8.3.3 Trade-Off Analysis of the Security Mechanisms

After completing the security analysis, we now turn our attention to trade-off analysis. The inputs to our trade-off analysis are the various subnets, static security level variables (SSLE), risk level variables (RL), security solution treatment level variables (SSTL), and the trade-off parameters (TOP), as shown in Figure 8.2.

Static Security Level (SSLE) Subnet

SSLE represents stakeholders' assessment of the value of system assets. There are always multiple stakeholders' viewpoints regarding system asset value, so the SSLE subnet topology includes variables that apply relative weight to a stakeholderâĂŹs assessment. The stakeholdersâĂŹ assessment of asset value and the stakeholders' weight are the observable nodes in the subnet. A decision node that represents the computation and its accompanying utility node, determines the influence of each stakeholder on the outcome of the subnet. For our example, the subnet computation leads to an SSLE value of *high*.

Risk Level (RL) Subnet

RL subnet incorporates the risks present in the initial design. All nodes are stochastic and are : (i) the average effort an attacker must use to launch a successful attack (METM), (ii) the mean time it takes for an attacker to launch an attack (MTTM), (iii) how often an attack will occur (MF), and (iv) the impact of an attack (MI). We derive the value for the risk variables MTTM, METM and MF directly from the result of the Alloy security analysis performed on the initial misuse model. The security analysis produced a counterexample for the passive man-in-the-middle attack, which is a simple attack, and one that requires little time or effort on the part of the attacker. The variable values we use, based on these results lead to an RL subnet computation distribution of RL.low = 0.1, RL.medium = 0.7, and RL.high = 0.2. This probability distribution function indicates that the risk level of the initial design is most likely *medium*.

Security Solution Treatment Level (SSTL) Subnet

The SSTL subnet contains variables relating to a security solution and how well it protects target assets. The SSTL subnet variables include the extent to which the solution provides security properties, its effect on Risk Level (RL) subnet variables METM, MTTM, MF and MI, and its cost. We model the cost as a subnet that combines implementation cost, maintenance cost and time to implement.

The security analysis did not produce a counterexample for the SRP security-treated misuse model. In the analysis we used a scope of 20, which for our example gives strong evidence that SRP protects the ACTIVE login sequence against active man-in-the-middle attacks. These results mean that the security effect (SE) is verified as being *high* and hence the variables SE on METM, SE on MTTM, SE on MI, and SE on MF, are all *high*. We define the cost of SRP as *medium*, because the code is not shipped with web clients as part of browsers, and thus it must be added to both clients and servers at the application level. The resulting computation is the following probability density function (pdf) for the target variable SSTL treatment level: SSTL.low = 0.0, SSTL.medium = 0.50, and SSTL.high = 0.50. The interpretation of this pdf is that it is just as likely that the treatment level is *medium* as *high* and the treatment level will never will be *low*.

The Alloy Analyzer security analysis of the SSL security-treated misuse model did produce a counterexample for an active man-in-the-middle attack. The counterexample demonstrates an attack that does not require more than medium attacker skill, little resources and time, so the solution effect is modeled as *low*. These results mean that all variables related to the effect of the security solution on an active man-in-the-middle attack are set to *low* (SE on METM, SE on MTTM, SE on MF, and SE on MI). We define the cost of SSL as *low* since the code is shipped with web clients as part of browsers.

For the passive version of the attack, the Alloy Analyzer did not produce a counterexample, so we infer that the SSL protocol preserves the security properties under this particular attack. Its effect on the risk variables MI, MF, METM, and MTTM is therefore *high*. This is in contrast to the active attack, where the security analysis showed that all variables are in the *low* state. The SSTL subnet is configured such that if all security effect variables are in the *low* state, both the solution effect and the resulting treatment level are in the *low* state, independent of the cost.

Trade-Off Parameters (TOP) Subnet

We identify three trade-off parameters of interest, namely, security acceptance criteria (SAC), time-to-market (TTM), and budget constraints. Since we need to produce a product in a small time, we define a value *short* for TTM. Our limited resource for incorporating a security solution to prevent man-in-the-middle attacks necessitates that we define a value of *medium* for budget. The SAC variable is actually an input node that receives input from an associated subnet which contains a node for each of the seven security possible security properties, namely, confidentiality, authenticity, integrity, accountability, availability, non-repudiation, and reliability. In our example, the first three security properties are equally relevant and the corresponding nodes are marked in the *high* state. The other properties are not applicable and are marked with *NA*.

We can also specify *priorities* which the designer can adjust when it is not possible to meet all the initial constraints. In our example, the following priorities are assigned: first priority is given to TTM, second priority is given to security requirements and third priority is given to budget. The priorities can be changed at any point of time and the analysis repeated.

Comparing SRP and SSL Security Solutions – Fitness Score

The Hugin tool computes each subnet, using all evidence entered into the topology, and propagates the results into the respective observable nodes in the top-level fitness score network, shown in Figure 8.2. The fitness score utility function uses a ranked-weight schema. Higher priority trade-off parameters are ranked with higher fitness scores so that factors other than security can be taken into account when deciding between alternative security solution designs. This schema also gives us the ability to easily change the importance of a trade-off parameter if project circumstances change and we need to put more emphasis on meeting a different project goal. The fitness score is thus a measure of the degree that a particular security solution meets the security, development, project and financial constraints of the project (specified in the TOP subnet).

The fitness score for the SRP security solution tells us that when the priorities are TTM, SAC and budget, the fitness of SRP in mitigating an active man-in-the-middle attack is more than 3 times more likely to be *high* than *low* and 1.6 times more likely to be *high* than *medium* (16% for low, 32% for medium, and 52% for high). Note that these results do not mean that the fitness score is *high* 52 times out of 100, but that our belief is that it will be *high* more than half the time within a particular time frame.

We compute the fitness score for SSL by changing the Security Solution Treatment Level (SSTL) variables in the top-level network in the BBN topology. The computation produces a fitness score of 23% for *low*, 23% for *medium*, and 54% for *high* for SSL in the presence of an active man-in-the-middle attack. For a passive man-in-the-middle attack, the result changes slightly and becomes 12% for *low*, 35% for *medium*, and 53% for *high*.

The above results imply that the fitness scores for SRP and SSL with the current trade-off parameter priorities differ by a small measure, so either one can be chosen. However, the situation changes if the priority of the trade-off parameters changes. If all emphasis is put on security requirements, meaning that the trade-off parameter SAC is given a priority of 100man-in-the-middle attack are taken into consideration, the fitness score changes to 20% for *low*, 75% for *medium*, and 5% for *high* for SRP and 0% for *low*, 90% for *medium*, and 10% for *high* for SSL. These results make sense, as the treatment level of SRP is higher than that of SSL in the context of the active man-in-the-middle attack. The fitness score is, however, still not completely in favor of SRP for two reasons. First, SRP involves a higher cost and time to market than SSL. Second, the risk level of the man-in-the-middle attack is most likely *medium*. SSL has a *low* treatment level for active attacks, but never a *low* treatment level in the context of passive man-in-the-middle attacks. For this the reason the fitness score of SSL turns out to be heavily ranked towards *medium* when both attack types are taken into consideration.

8.4 Conclusion and Future Work

Ad-hoc approaches for developing secure systems may result in security breaches. We propose an AORDD methodology for designing secure systems. The first step is to perform a risk assessment to identify the attacks on the system and evaluate how the assets of the system can be compromised. In order to protect against these attacks, security solutions must be methodically incorporated into the system. The resulting system is then formally evaluated to give assurance that it is indeed secure. Multiple security solutions are often effective in protecting against a given attack, so designers must identify and integrate the one that is most suitable for the application.

In this work, we focussed the case for a single attack. However, in reality, there are multiple attacks and multiple security solutions must be incorporated. Moreover, incorporating a security solution should not open up new vulnerabilities. Designers can continually augment system designs by composing additional security solutions to mitigate additional attacks. They can then compose multiple attack models with these system models, and analyze them with the Alloy Analyzer. However, this approach can be cumbersome for designers, so we hope to provide an easier approach for handling multiple attacks. In this respect, we are currently investigating techniques that formalize the dependencies between the different types of attacks and security solutions. Such formalization will allow us to group attacks and security solutions. This, in turn, will facilitate minimizing the time required for security analysis.

Chapter 9

Conclusion

Pervasive computing applications have some unique constraints that preclude the use of traditional security policies and mechanisms for protecting such applications. We investigated the security requirements of pervasive computing applications and proposed several solutions. Our first contribution is proposing new access control models that use contextual information, namely, location and time, to provide access control. The model has several features which may interact in subtle ways. We showed how such a model can be used for real-world application and analyzed to ensure that access control breaches do not occur. We also provided a graph-theoretic semantics that helps the user in visualizing the policies and allows the use of graph algorithms to detect problems with the specification.

Pervasive computing applications often involve interaction among entities not all of whom are trusted to the same extent. We proposed a formal model of trust, based on subjective logic, that allows one to argue about the trustworthiness of entities. The model is also able to quantify uncertainty with respect to trust, which is inherent in pervasive computing applications. We also demonstrated how the trust model can be used to make access control decisions and how to transmit data reliably through a sensor network.

Pervasive computing applications typically collect contextual information, some of which is sensitive in nature. Towards this end, we proposed new models that allow for controlled data dissemination. Specifically, we proposed new algorithms for preserving location privacy and we also developed metrics with which to compare different privacy algorithms. Data availability also plays an important part in pervasive applications. We proposed techniques using which data can be efficiently obtained in pervasive computing applications.

Pervasive computing applications operate in a heterogeneous environment where all the nodes in the network may not have the same computation and communication capabilities. We showed how to calculate the risk in such environments. We also demonstrated how to do optimal security provisioning when all attacks cannot be prevented due to resource constraints.

Security cannot be added as an afterthought to pervasive computing applications. Security issues must be addressed at the design phase. We showed how aspect-oriented risk-driven methodology can be used for designing pervasive computing applications, how can we get assurance that such a design is correct, and how do we do trade-off among various security properties when all properties cannot be satisfied due to various constraints.

This report highlights the major contributions of the project. Some of the results have not been included in this report to save space, though they have briefly mentioned in passing. These can be found in our publications, the complete list of which has been included in the report.

The project has helped us identify a number of open research problems that we plan to address in future projects. Our future work involves identifying, formalizing, and quantifying each security attribute (confidentiality, integrity, availability, non-repudiation, etc.) in pervasive computing applications. This involves understanding each attribute, identifying the invariant properties of these attributes, and the relationship among the attributes. We also plan to understand security solutions in more details. For instance, we need to know what attributes are ensured by each solution and to what extent. We also need to investigate composability properties of security solutions. For example, it is possible that a security mechanism preserves some attribute when used in isolation, but may not do so when used in conjunction with another security solution. We plan to formalize vulnerabilities and attacks and understand how attacks impact the attributes. An attack is possible if some invariant property of a security attribute is destroyed. Enumerating all possible ways in which such invariant properties can be destroyed will allow us to predict future attacks as well. Once the inherent properties of attributes, attacks, and solutions are understood, we can provide better protection against known and unknown attacks.

Bibliography

[1] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. UML2Alloy: A Challenging Model Transformation. In *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 436–450, Nashville, TN, U.S.A., September 2007.

[2] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. On Challenges of Model Transformation from UML to Alloy. *Software and Systems Modeling*, 9(1):69 – 86, January 2010.

[3] ArgoUML. ArgoUML website, 2009. `http://argouml.tigris.org` [cited September 2010].

[4] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A Temporal Role-Based Access Control Model. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 21–30, Berlin, Germany, July 2000.

[5] C. Bettini, X.S. Wang, and S. Jajodia. Protecting Privacy Against Location-Based Personal Identification. In *Proceedings of the 2nd VLDB Workshop on Secure Data Management*, volume 3674 of *Lecture Notes in Computer Science*, pages 185–199, Trondheim, Norway, 2005.

[6] S. A. Butler. Security Attribute Evaluation Method: A Cost-Benefit Approach. In *Proceedings of the 24th International Conference on Software Engineering*, pages 232–240, Orlando, FL, U.S.A., May 2002.

[7] S. A. Butler and P. Fischbeck. Multi-attribute Risk Assessment. In *Proceedings of the Symposium on Requirements Engineering For Information Security*, Raleigh, NC, U.S.A., October 2002.

[8] L. Chen and J. Crampton. On Spatio-Temporal Constraints and Inheritance in Role-Based Access Control. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, pages 205–216, Tokyo, Japan, March 2008.

[9] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley. Optimal Security Hardening Using Multi-objective Optimization on Attack Tree Models of Networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 204–213, Alexandria, VA, USA, October-November 2007.

[10] R. Dewri, I. Ray, N. Poolsappasit, and D. Whitley. Optimal Security Hardening on Attack Tree Models of Networks: A Cost-Benefit Analysis. *International Journal of Information Security*, Under review.

[11] R. Dewri, I. Ray, I. Ray, and D. Whitley. Security Provisioning in Pervasive Environments Using Multi-objective Optimization. In *Proceedings of the 13th European Symposium on Research in Computer Security*, number 5283 in Lecture Notes in Computer Science, pages 349–363, Malaga, Spain, October 2008.

[12] R. Dewri, I. Ray, I. Ray, and D. Whitley. On the Formation of Historically k-Anonymous Anonymity Sets in a Continuous LBS. In *Proceedings of the 6th International ICST Conference on Security and Privacy in Communication Networks*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, Singapore, September 2010.

[13] T. Dimitrakos, B. Ritchie, D. Raptis, J. Ø. Aagedal, F. d. Braber, K. Stølen, and S. H. Houmb. Integrating Model-based Security Risk Management into eBusiness Systems Development: The CORAS Approach. In *Proceedings of the 2nd IFIP Conference on E-Commerce, E-Business, E-Government*, pages 159–175, Lisbon, Portugal, 2002.

[14] B. Gedik and L. Liu. Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms. *IEEE Transactions on Mobile Computing*, 7(1):1 – 18, January 2008.

[15] G. Georg, J. Bieman, and R. B. France. Using Alloy and UML/OCL to Specify Run-Time Configuration Management: A Case Study. In *Proceedings of the Workshop of pUML*, volume 7 of *LNI*, pages 128–141, October 2001.

[16] B. A. Gran. The Use of Bayesian Belief Networks for Combining Disparate Sources of Information in the Safety Assessment of Software Based System. Technical report, Norwegian University of Science and Technology, Trondheim, Norway, 2002.

[17] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services*, pages 31–42, San Francisco, CA, 2003.

[18] HUGIN. Hugin Expert A/S, 2007.

[19] ISO 14508. Common Criteria for Information Technology Security Evaluation Version 3.1 Revision 2, 2007.

[20] ISO 14508-4. Common Methodology for Information Technology Security Evaluation Version 3.1 Revision 2, 2007.

[21] ISO/IEC 13335-5. Information Technology – Guidelines for Management of IT Security, 2001.

[22] D. Jackson. Automating First-Order Relational Logic. In *Proceedings of the 8th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 130–139, San Diego, CA, U.S.A., November 2000.

[23] D. Jackson. Micromodels of Software: Lightweight Modelling and Analysis with Alloy, 2002. `http://alloy.mit.edu/alloy2website/reference-manual.pdf` [cited September 2010].

[24] D. Jackson. Alloy 3.0 Reference Manual, 2004. `http://alloy.mit.edu/reference-manual.pdf` [cited September 2010].

[25] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, Cambridge, MA, U.S.A, 2006.

[26] A. Jøsang. Artificial Reasoning with Subjective Logic. In *Proceedings of the 2nd Australian Workshop on Commonsense Reasoning*, Perth, Australia, December 1997.

[27] A. Jøsang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, June 2001.

[28] A. Jøsang, E. Gray, and M. Kinateder. Simplification and Analysis of Transitive Trust Networks. *Web Intelligence and Agent Systems Journal*, 4(2):139–161, 2006.

[29] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A Generalized Temporal Role-Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, January 2005.

[30] X. Liu and G. Schrack. Encoding and Decoding the Hilbert Order. *Software Practice and Experience*, 26(12):1335 – 1346, December 1996.

[31] S. Mascetti, C. Bettini, X. S. Wang, D. Freni, and S. Jajodia. ProvidentHider: An Algorithm to Preserve Historical k-Anonymity in LBS. In *Proceedings of the 10th International Conference on Mobile Data Management*, pages 172–181, Taipei, Taiwan, May 2009.

[32] J. Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2), 1950.

[33] OMG. Unified Modeling Language: Superstructure. Version 2.0. Document id: formal/05-07-04. `http://www.omg.org`.

[34] OMG. Unified Modeling Language: Superstructure Version 2.1.2, 2002. Document Id: formal/07-11-02.

[35] OMG. XML Metadata Interchange Version 2.0, 2005. Document Id: formal/05-05-01.

[36] OMG. Meta Object Facility Core v. 2.0, 2006. Document Id: formal/06-01-01. `http://www.omg.org`.

[37] OMG. Object Constraint Language Version 2.0, 2006. Document Id: formal/06-05-01. `http://www.omg.org`.

[38] N. Poolsappasit, R. Dewri, and I. Ray. Dynamic Security Risk Management Using Bayesian Attack Graphs. *IEEE Transactions on Dependable and Secure Computing*, Under Review.

[39] I. Ray, N. Poolsappasit, and R. Dewri. An Opinion Model for Evaluating Malicious Activities in Pervasive Computing Systems. In *Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 5094 of *Lecture Notes in Computer Science*, pages 297–312, London, U.K., 2008.

[40] I. Ray, I. Ray, and S. Chakraborty. An Interoperable Context-Sensitive Model of Trust. *Journal of Intelligent Information Systems*, 32(1), 2009.

[41] I. Ray, I. Ray, and S. Chakraborty. A Context-Aware Model of Trust for Facilitating Secure Ad Hoc Collaborations. In Z. Yan, editor, *Trust Modeling and Management in Digital Environments: From Social Concept to System Development*. IGI Global, 2010.

[42] I. Ray and M. Toahchoodee. A Spatio-temporal Role-Based Access Control Model. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 211–226, Redondo Beach, U.S.A., July 2007.

[43] I. Ray and M. Toahchoodee. A Spatio-Temporal Access Control Model Supporting Delegation for Pervasive Computing Applications. In *Proceedings of the 5th International Conference on Trust, Privacy & Security in Digital Business*, pages 48–58, Turin, Italy, September 2008.

[44] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universitaet Bremen, 2002. Logos Verlag, Berlin, BISS Monographs, No. 14.

[45] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[46] R. Simon and M. E. Zurko. Separation of Duty in Role-based Environments. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 183–194, Rockport, MA, USA, June 1997.

[47] K. Stølen, F. d. Braber, R. Fredriksen, B. A. Gran, S. H. Houmb, Y. C. Stamatiou, and J. Ø. Aagedal. Model-based Risk Assessment in a Component-Based Software Engineering Process – Using the CORAS Approach to Identify Security Risks. In *Business Component-Based Software Engineering*. Springer, 2002.

[48] M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *Formal Techniques for Networked and Distributed Systems - FORTE 2 003*, volume 2767 of *Lecture Notes in Computer Science*, pages 240–256, 2003.

[49] K. W. Tan, Y. Lin, and K. Mouratidis. Spatial Cloaking Revisited: Distinguishing Information Leakage from Anonymity. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*, volume 5644 of *Lecture Notes in Computer Science*, pages 117–134, Aalborg, Denmark, July 2009.

[50] M. Toahchoodee and I. Ray. On the Formal Analysis of a Spatio-Temporal Role-Based Access Co ntrol Model. In *Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 17–32, London, U.K., July 2008.

[51] M. Toahchoodee and I. Ray. Using Alloy to Analyze a Spatio-Temporal Access Control Model S upporting Delegation. *IET Information Security*, 3(3):75 – 127, September 2009.

[52] M. Toahchoodee and I. Ray. On the Formalization and Analysis of a Spatio-Temporal Role-Based Access Control Model. *Journal of Computer Security*, 2010. To appear.

[53] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar. Ensuring Spatio-Temporal Access Control for Real-World Applications. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, pages 13–22, Stresa, Italy, June 2009.

[54] M. Toahchoodee, I. Ray, and R. M. McConnell. Using Graph Theory to Represent a Spatio-Temporal Role-Based Access Control Model. *International Journal of Next Generation Computing*. Under Review.

[55] Transport Layer Security Working Group. SSL 3.0 Specification, 1996.

[56] T. D. Wu. The Secure Remote Password Protocol. In *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, USA, 1998.

[57] J. Zao, H. Wee, J. Chu, and D. Jackson. RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis, 2002. `http://alloy.mit.edu/publications.php` [cited September 2010].

REPORT DOCUMENTATION PAGE

1. REPORT DATE 10-10-2010	2. REPORT TYPE Final Report	3. DATES COVERED Dec 2006 - May 2010

4. TITLE AND SUBTITLE
Addressing Security Challenges in Pervasive Computing Applications

5a. CONTRACT NUMBER
FA 9550-07-1-0042

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

6. AUTHOR(S)
Ray, Indrakshi
Ray, Indrajit

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Colorado State University
Room 408 USC
601 S. Howes Street
Fort Collins CO 80523

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. Robert Herklotz
AFOSR
Suite 325 Room 3112
875 N. Randolph Street
Arlington VA 22203-1768

10. SPONSOR/MONITOR'S ACRONYM(S)
DOD-USAF-Air Force

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-OSR-VA-TR-2012-0150

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A: Approved for Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Pervasive computing provides services that use knowledge about the operating environment. The very knowledge that enables an application to provide better services may, however, be misused, causing security and privacy breaches. Uncontrolled disclosure of information or unconstrained interaction among entities may have serious consequences. Traditional security policies and mechanisms are inadequate because these applications are dynamic and are executed in resource constrained environment. This work proposed new access control models and trust models for use in pervasive computing applications. The work also investigated risk management and security provisioning for pervasive environments when limited resources are available. Controlled data dissemination techniques were also proposed as a part of this work. Finally, the work also discussed methodologies for designing secure pervasive computing applications.

15. SUBJECT TERMS
Pervasive computing, access control models, trust models, secure design, controlled data dissemination

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Indrakshi Ray
Unclassified	Unclassified	Unclassified	SAR	108	19b. TELEPHONE NUMBER 1-970-491-7986

20120918157